Eine Formularerweiterung für XSL-FO

Studienarbeit
von
Florian Hecht
8. Dezember 2004

Institut für Programmstrukturen und Datenorganisation (IPD)
Lehrstuhl Prof. Dr. W. F. Tichy
Universität Karlsruhe (TH)
Betreuer: Dipl. Inf. Tom Gelhausen

Inhaltsverzeichnis

Eine Formularerweiterung für XSL-FO

Kurzfassung. In dieser Studienarbeit wurde die bestehende Layoutsprache XSL-FO um die Möglichkeit erweitert, Formulare explizit zu definieren. Dadurch können aus solchen Dokumenten sowohl papierbasierte als auch elektronische Formulare erzeugt werden. Zur Umsetztung wurde der FO-Prozessor von apache.org erweitert, so dass er die Erweiterung verarbeiten kann. Mit dem veränderten Prozessor können PDF-Dateien mit Formularelementen generiert werden, die wie HTML Formulare abgeschickt werden können.

1 Einleitung

1.1 Motivation

In vielen Firmen und Betrieben gibt es eine Vielzahl von spezialisierten Formularen für Arbeitsstunden, Reisekosten oder andere Anträgen. Oft müssen die ausgefüllten Formulare durch mehrere Hände bis sie kontrolliert, bestätigt und anerkannt sind. Dieser Ablauf könnte beschleunigt werden, wenn die Formulare digital bearbeitet und mittels elektronischer Post verschickt werden könnten. Es würde ebenso Papier gespart werden. Jedoch müssen die Formulare erst einmal entworfen werden und in manchen Fällen sind Papierformulare nicht zu ersetzen. Es wäre daher vorteilhaft die Formulargestaltung so zu nutzen, dass sowohl digitale als auch papierbasierte Formulare daraus erzeugt werden können, so dass bei jeder Nutzung der Formulare die am besten geeignete Version verwendet werden kann.

Formulare werden in der Regel mit einem Layoutsystem entworfen. Die meisten Layoutsysteme sind aber nicht auf Formulare spezialisiert. Sie erkennen einen Kasten auf dem Formular nicht als spezielles Feld, in das Informationen eingetragen werden. Für das Layoutsystem handelt es sich nur einen Kasten im Layout. Um aber aus einer Formularbeschreibung ein digitales Formular zu erzeugen, muss das System erkennen können welche Kasten Eingabefelder sind und welche nur zum Layout gehören. Will man nun ein System schaffen, das beide Varianten von Formularen erzeugen kann, so muss man in diesem System Eingabefelder spezifiziert können. Es liegt nahe ein bestehendes Layoutsystem um die Spezifikation von Formularelementen zu erweitern und dann die Applikation, die die Formulare auf Papier umsetzt, so zu verändern, dass sie auch digitale Formulare erzeugen kann.

XSL ist ein solches Layoutsystem und XSL-FO die darin enthaltene Spezifikationssprache für Layout. Formularelemente können hierin nicht angegeben werden. Um zu verstehen welche Bedeutung eine Formularerweiterung für XSL-FO hat, soll klar gemacht werden, was XSL-FO genau ist. Es soll ebenso genauer abgegrenzt werden, was wir unter einem Formular verstehen, um die Grundlagen einer Formularerweiterung für XSL-FO und ihrer Entwicklung darzulegen.

1.2 XSL

XSL (eXtensible Stylesheet Language) ist eine Spezifikation für die Umwandlung von XML-Dateien in eine formatierte Darstellung [1]. Die Spezifikation besteht aus zwei Teilen: XSLT und XSL-FO.

XSL-FO (eXtensible Stylesheet Language – Formating Objects) ist eine Layout-Beschreibungssprache, die auf XML basiert. Sie wurde vom World Wide Web Consortium in der W3C Style Activity entwickelt und ist Teil der Extensible Stylesheet Language Family XSL. XSL-FO erreichte am 15. Oktober 2001 den 1.0 Status zusammen mit seinem bekannteren „großen“ Bruder XSLT (XSL Transformations). Die beiden Standards gehören zur selben Familie, da sie beide einen ganz bestimmten Zweck in einer „Styling-Pipeline“ erfüllen. Hierbei beschreibt XSLT generelle Transformationen von einem XML-Dokument in eine anderes. Die Transformation wird wiederum in einem XML-Dokument in einer deklarativen Programmiersprache spezifiziert. Eine sehr gute Einführung und Referenz findet sich in [2].

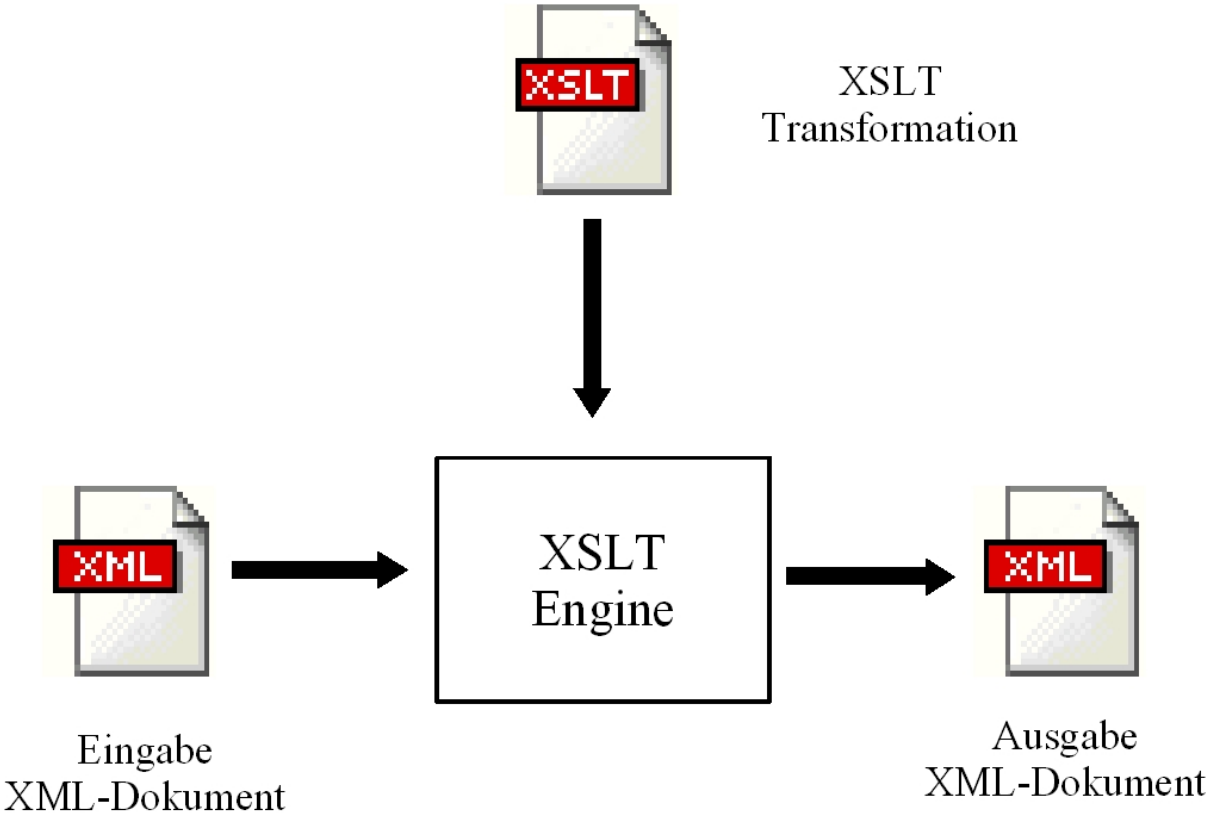

Fig. 1. Ein XSLT-Prozessor nimmt als Eingabe ein XML-Dokument und ein XSLT-Dokument, das die Transformation beschreibt. Als Ausgabe wird ein neues XML-Dokument erzeugt

XSLT kann zwar generell XML in XML transformieren, doch sowohl der ursprüngliche Zweck als auch die häufigste Verwendung besteht darin, XML Rohdaten beim Transformieren mit „Style“ zu versehen. Die häufigste Anwendung dessen ist die Transformation von XML nach HTML.[1] Dabei werden die Daten aus der/den XML-Datei(en) in Tabellen oder Listen im HTML-Format wiedergegeben und dabei Farben, Schriftarten, Abstände oder Überschriften festgelegt. Die hauptsächliche Nutzung dafür liegt darin, dass im Moment Webseiten als generelles Ausgabeformat dienen, da auf jedem Computer oder PDA Browser vorhanden sind die diese anzeigen können.

1 genauer gesagt XHTML. Nur XHTML ist wirklich XML konform, HTML nicht ganz.

Als die XSL Familie entworfen wurde, wurden jedoch auch Printmedien berücksichtigt und deshalb wurde XSL-FO als eine Spezifikationsform für professionelles Layout entwickelt. Ähnlich wie HTML (ohne Berücksichtigung von Cascading Style Sheets) enthält FO sowohl Daten als auch das fertige Layout. Der wichtige Unterschied zwischen HTML und FO besteht darin, dass HTML vom Browser interpretiert wird und je nach Größe des Browser-Fensters oder des Bildschirms dargestellt wird. FO ist jedoch eine Spezifikation für festes Layout und wird deswegen nicht interpretiert. Wie die Angaben in einem FO-Dokument vom jeweiligen FO-Prozessor zu verarbeiten sind, sind im XSL-FO-Standard genau geregelt. Eine Interpretation ist nicht nötig, da das Ausgabemedium, d.h. die Seitendimensionen, im FO-Dokumnet mitspezifiziert werden. Das FO-Format selbst ist geräteunabhängig. Es ist also ein weitere Schritt („Rendern“) nötig, der ein FO-Dokument in seine fertig gelayoutete Form überführt. Dieser Schritt wird vom oben schon erwähnten FO-Prozessor durchgeführt. Er setzt die Anweisungen des FO-Dokuments dem Standard entsprechend um und erzeugt eine Ausgabe in einem (unter Umständen) nicht mehr geräteunabhängiges Format. Dies können direkte Druckeranweisungen oder auch eine PostScript/PCL/PDF-Datei sein. Das direkte Darstellen auf einem Bildschirm ist natürlich ebenfalls möglich.

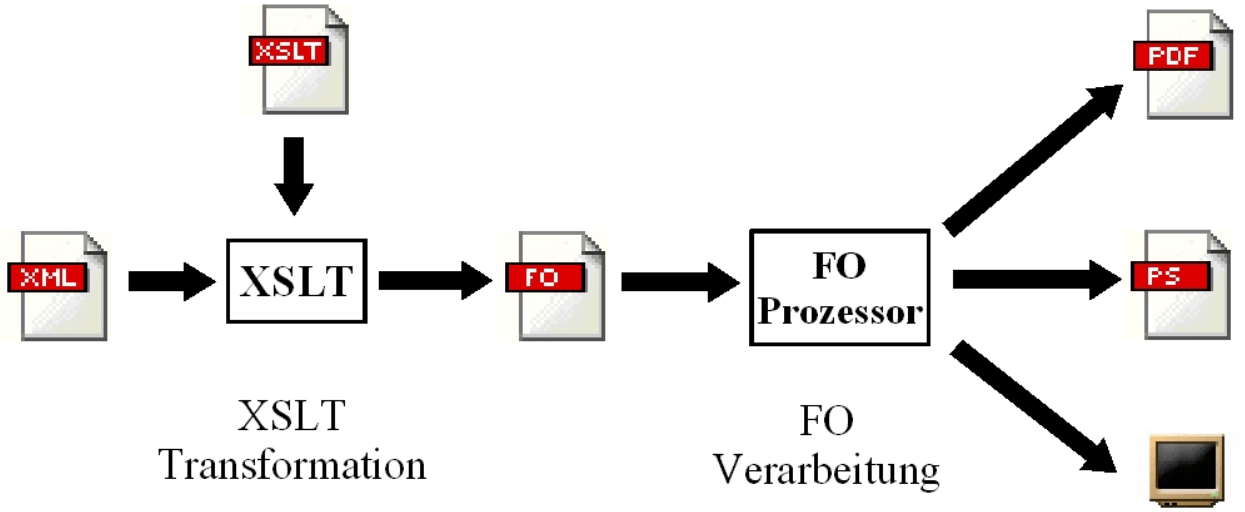

Fig. 2. Zu erst werden die XML-Daten mittels XSLT in ein FO-Dokument transformiert. Dann können aus diesm FO-Dokument mit einem FO-Prozessor verschiedene Ausgabeformate erzeugt werden.

Wenn man nach anderen Systemen sucht, die das gleiche Leisten, so findet man das TeX-System. Seit über zwei Jahrzehnten in Benutzung, ist es wohl die Referenz was Layoutsysteme für Bücher und Artikel im wissenschaftlichen Umfeld angeht. Gerade das es schon so lange in Benutzung ist, zeigt, dass es für die gegebenen Aufgaben taugt und jedes neue System muss sich daran messen. XSL-FO hat bis jetzt noch keine weite Verbreitung gefunden, vor allem da es immer noch im Schatten des des bekannteren TeX-Systems steht. Außerdem ist das TeX-System als ganzes wesentlich umfangreicher in seinen Möglichkeiten als XSL-FO. Mit XSL-FO kann man den Textfluß steuern und sowohl Blöcke als auch Text formatieren, jedoch gehören mathematische Formelkonstruktion und ein Makrosystem nicht zum Umfang von XSL-FO. Dabei tragen gerade diese Teile des TeX-Sytems zur Popularität bei. XSL-FO hat jedoch auch ein paar Vorteile zu bieten, die durch die XML Herkunft entstehen. XSL-FO Systeme können auf bereits entwickelten XML-Infrastrukturen aufbauen, sowohl was das

Verarbeiten von FO Dokumenten angeht als auch bei der Verifizierung über Schema- oder DTD-Spezifikationen. XSL-FO hat einen eigenen XML-Namensraum, (Namespace: http://www.w3.org/1999/XSL/Format) und lässt sich deshalb auch in andere XML-Dokumente einbetten und man kann auch leicht Erweiterungen über einen weiteren Namensraum in ein FO-Dokument einbinden, was auch bei der später vorgestellten Erweiterung so gemacht wurde. Die enge Verzahnung mit XSLT ist natürlich ein weiterer Pluspunkt. XSL-FO ist zwar (noch) nicht so mächtig wie TeX, doch bereits jetzt sehr gut in eine auf XML basierende Pipeline eingebunden, was vor allem für automatisch generierte Dokumente von Vorteil ist.

1.3 Formulare

Formulare dienen dazu von einer Person, dem Ausfüller, Information in geordneter Form zu erhalten. Geordnet deswegen weil der Ort an dem eine bestimmte Information „abgelegt“ wird genau bestimmt ist. Formulare dienen auch dazu diese Informationen zu speichern, in dem die Formulare aufbewahrt werden, oder um diese Informationen weiter zu verarbeiten, in dem eine andere Person die Informationen ausließt und damit weiter arbeitet.

Ein Formular befindet sich wie normaler Text auf einer oder mehreren Seiten Papier. Es hat aber ein paar Elemente, die es von einem gewöhnlichen Text abheben. Es handelt sich hierbei vor allem um die vorgegebenen Lücken für die einzutragenden Daten. Ob es sich um einen Kasten mit oder ohne Unterteilung für einzelne Buchstaben oder um Kästchen zum Ankreuzen, Ausmalen oder Durchstreichen handelt, es sind gerade diese Elemente die ein Formular „interaktiv“ machen und somit für seinen Zweck unerlässlich sind. Wie bereits angedeutet, gibt es mehrere Möglichkeiten ein und die selbe Information vom Ausfüller zu erhalten. Die Flächen für Angaben, die in Form eines „freien“ Textes gesammelt werden wie Name, Straße oder Stadt, können auf verschiedene Art und Weisen angezeigt werden. Durch eine simple Linie auf der man schreiben soll, durch eine gepunktete Linie oder auch durch einen Kasten. Für die maschinelle Verarbeitung handschriftlich ausgefüllter Formulare bietet es sich auch an für die einzelnen Buchstaben der Texte einzelne Kästchen anzugeben um dem Schrifterkenner die Arbeit zu vereinfachen. Dazu gehört natürlich auch die Bitte mit Druckbuchstaben zu schreiben. Bei solchen Texten wird die Freiheit des Ausfüllers aus praktischen Gründen eingeschränkt um die Verarbeitung zu beschleunigen. Wenn die Eingabe an sich schon sehr beschränkt ist so macht es auch Sinn den Platz für die Eingabe einzuschränken, dies bietet sich zum Beispiel bei Datumsangaben oder bei Kontonummer an, da die Anzahl der Stellen bereits im Voraus bekannt ist, ebenso natürlich bei anderen Identifikationsnummern. Eine weitere Einschränkung kann vorgenommen werden, wenn nur nach Ja oder Nein gefragt wird. Hier wird dem Ausfüller die Möglichkeit geboten durch eine einzige Markierung eine Aussage zu machen. Wobei man nicht markiert normalerweise als Nein auf die gegebene Frage wertet und markiert eben als Ja. Für Markierung gibt es mehrere Möglichkeiten: Ein Kästchen zum Ankreuzen oder um ein Häkchen zu machen. Oder ein Kreis, der zum Markieren ausgefüllt wird. Manchmal ist es von Vorteil sowohl eine Markierungsmöglichkeit für Ja als auch eine für Nein anzubieten (Zwei Kästchen, zwei Kreise, etc.) um auch die Möglichkeit des Nichtbeantwortens zu bieten (oder des Falschbeantwortens, indem sowohl Ja als auch Nein markiert wird). Gruppiert man mehrere solche einzelnen Ja/Nein Möglichkeiten, so kann man 1-aus-n oder k-aus-n Auswahlmöglichkeiten anbieten. Es muss dazu dem Ausfüller nur klargemachen werden, das diese einzelnen Markierung zusammen eine Auswahl darstellen, dies kann durch entsprechende Umrahmung oder Erklärungen gemacht werden.

Aus diesem Schema fällt die Unterschrift unter einem Formular heraus, da nicht der eigentlich

Text, der Name, interessant ist, sondern die Unterschrift an sich. Die rechtliche Bindung bzw. die Verifizierbarkeit wir nicht durch den Namen oder das Kürzel repräsentiert sondern durch die Gesamtheit der Signatur und ist unmöglich zu Abstrahieren.

Der Trend zu elektronischen bzw. digitalen Formularen ist unverkennbar. Lieferfirmen füllen Formulare am Rechner aus und lassen sie vom Empfänger vor Ort auf einem Touch-Display unterschreiben. Anmeldungen und Bestellungen über Webseiten sind auf dem besten Weg zum Standard zu werden. Die inhaltlichen Hinweise sind natürlich immer noch von Nöten, denn ein Eingabefeld ohne jegliche Beschriftung ist immer noch nutzlos, jedoch ist ein ausgefülltes Eingabefeld immer lesbar, kann leicht verbessert werden und fast noch wichtiger es liegt bereits digital vor, bereit zum Versand an eine verarbeitende Applikation. Die Formulare, die mit HTML konstruiert werden, können fast alles darstellen was auch ein Papierformular kann (außer Unterschriften), zusätzlich gibt es noch Alternativen für 1-aus-n und k-aus-n Auswahlmöglichkeiten in Form von Listauswahlfeldern (Comboboxes) und Mehrfachlistenauswahlfeldern (Listboxes), die in der bekannten Form schwer bis gar nicht auf Papier umzusetzen sind. Das Abschicken geschieht nicht wie bei Papierformularen, durch Übergabe an den Sachbearbeiter, sondern durch drücken des Sendeknopfs, ebenso sind bei elektronischen Formularen auch Resetknöpfe möglich. Das einzige mit dem sich elektronische Formulare noch schwer tun sind Unterschriften. Dieses Problem ist bis heute nicht vollständig gelöst, obwohl Signaturen als digitale Unterschriften im kommen sind, haben sie die Akzeptanz ihrer Vorbilder noch nicht erreicht.

1.4 Erweiterung

Es wurde eine Erweiterung entwickelt, die es ermöglicht Formularelemente in einem FO-Dokument zu spezifizieren. Dabei wurde von vornherein auf digitale Formulare hin entwickelt. Es können natürlich auch konventionelle Formulare gestaltet werden, man würde dabei nur einen Teil der Erweiterung nutzen. Die Einflüsse, die die Erweiterung bestimmt und geformt haben, werden in Kapitel 2 behandelt. Es werden die auf XML-basierten Formular-/Dialog-Beschreibungssprachen XAML, Views für ROTOR und XForms betrachtet. Ebenso werden die Darstellungsformen für Formulare über HTML-Forms, PDF und AWT analysiert und die Fähigkeiten der verschiedenen Systeme verglichen. In Kapitel 3 wird dann die Erweiterung vorgestellt, die FEX getauft wurde, für FormEXtension. Es werden die Entwurfsprinzipien und die Einbettung in XSL-FO beschrieben, so wie die dazu nötige XML-Syntax samt ihrer Semantik. In Kapitel 4 werden die Erfahrungen mit der Erweiterung des FO-Prozessors von apache.org (genannt FOP) um die Formularerweiterung beschrieben, so wie den generellen Verlauf der FO-Verarbeitung. Der PDF-Renderer des FOP wurde um die Fähigkeit erweitert PDF-Dateien mit Formularen zu generieren. Kapitel 5 soll die Erfahrungen dieses gesamten Projekts zusammenfassen und Möglichkeiten für eine Fortsetzung zeigen.

2 Systeme zur Formularbeschreibung

Um verschiedene Systeme, die Formulare beschreiben können, vergleichen zu können braucht man ein Maß. Für Formularsysteme, vor allem für digitale Formulare, sind HTML-Formulare eine gute Messlatte, da sie, wie in der Einleitung bereits erwähnt, ein Defacto-Standard sind. Jedes Formularsystem das seinen Zweck erfüllen will muss zumindest die Fähigkeiten von HTML-Formularen besitzen. Weitere Fähigkeiten sind dann die, die das jeweilige System von den anderen abhebt. Um also die noch folgenden Systeme bewerten zu können sollen als erstes HTML-Formulare betrachtet werden. HTML-Formulare sind in doppelter Weise interessant. Auf der einen Seite steckt im HTML-Code die Beschreibung der Formulare in XML-Syntax

(nicht immer XML-konform) vor, so dass man das „Wie“ der Beschreibung betrachten kann. Auf der anderen Seite sind HTML-Formulare oft auch das Ziel anderer Prozesse zur Erzeugung von Formularen, z.B. php, ASP oder über MS-Word. Diese sind zwar nicht auf Formulare spezialisiert, erzeugen jedoch HTML als Ausgabeformat. Somit muss die innere Darstellung der Formulare in HTML-Code konvertiert werden. Dadurch kann das „Was“, der möglichen Formularelemente in HTML-Seiten, betrachtet werden.

2.1 HTML Formulare

Die Syntax für HTML-Formulare ist ein kleiner Teil der gesamten HTML-Syntax. Innerhalb eines HTML-Dokuments können beliebig viele Formulare definiert werden, dies geschieht über den *form* Tag. Innerhalb dieses Tags können wieder gewöhnliche HTML Ausdrücke stehen, die einzigen Elemente die das *form* Tag betreffen sind die Formularelemente. Alle Formularelemente die innerhalb eines *form* Tags definiert werden, gehören zu diesem Formular und der Inhalt dieser Elemente wird übermittelt sobald dieses Formular abgeschickt wird. Eine Schachtelung von Formularen ist nicht vorgesehen, es können auch keine Formularelemente von anderen Formularen zwischen geschoben werden. Im *form* tag wird bereits festgelegt an wen die Daten übermittelt werden. Es gibt Schaltflächen (buttons), Eingabefelder (editfields), mehrzeilige Eingabefelder (multiline editfields), Auswahlfelder (check boxes), Umschaltfelder (radio buttons), Mehrfachlistenauswahlfelder (list boxes), Listenauswahlfelder (combo boxes) und versteckte Werte (hidden values). Die Schaltflächen können zum Auslösen des Absendens (submit), zum Auslösen des Rücksetzens (reset) oder zum Auslösen beliebiger JavaScript (ECMAScript) Funktionen genutz werden. Bei Eingabefeldern kann auch ein Passwortmodus aktiviert werden, der die Eingabe unlesbar macht, damit Zuschauer Passwörter nicht mitlesen können. Mehrzeilige Eingabefelder werden benutzt, wenn eine längere Texteingabe, wie z.B. bei Kommentaren oder bei einem Eintrag in eine Webforum gewünscht ist, also mehrere Zeilen an Text notwendig ist. Auswahlfelder und Umschaltfelder entsprechen den Kästchen zum Ankreuzen, wobei Umschaltfelder immer in Gruppen vorkommen, von denen jeweils nur eines ausgewählt ist, es handelt sich hierbei also um eine 1-aus-n Auswahlmöglichkeit. Mehrfachlistenauswahlfelder stellen eine Auswahl von Möglichkeiten dar, von denen eine oder auch mehrere Selektiert werden können (k-aus-n). Listenauswahlfelder sind Eingabeelemente die eine platzsparende 1-aus-n Auswahl darstellen, bei der nur die momentan ausgewählte Möglichkeit angezeigt wird und die anderen sich erst bei einem Klick entfalten. Versteckte Werte dienen dazu Daten mit dem Formular zu verschicken, die vom Nutzer nicht verändert, ja nicht einmal gesehen werden können. Meistens werden sie genutzt um Informationen in aufeinander folgenden Formularen in verschiedenen HTML-Dokumenten zu übertragen, dabei werden benötigten Informationen im jeweils nächsten Formular als versteckte Werte untergebracht. Dadurch braucht der Server keine Daten über den momentanen Ausfüllvorgang serverseitig speichern, sondern bekommt mit jedem neu übertragenen Formular wieder die gesamten Informationen. HTML-Formulare stellen das notwendige an Eingabemöglichkeiten zur Verfügung. HTML-Formulare sind auch schon relative lange bei HTML mit dabei, sie haben sich jedoch kaum weiterentwickelt. Es gibt auch Dinge die wünschenswert wären, jedoch wohl nie in HTML-Formularen möglich sein werden, wie z.B. Typisierung der Eingaben und Validierung, so wie Manipulation von mehreren Datensätzen über ein und das selbe Formular. An dieser Stelle setzt der XForms Standard an.

2.2 XForms

XFoms ist eine Initiative des W3C und soll die Nachfolge von HTML-Formularen antreten, da es nach Aussage der Autoren „die nächste Generation der Formulare für das Web darstellt“.

Dies versucht der Standard nicht durch aufblähen seiner Formularfähigkeiten durch neue Steuerelemente, die es bei HTML-Formularen noch nicht gibt, sondern durch eine Umstrukturierung der Formulare in drei Teile zu erreichen: Ein Modell der auszufüllenden Nachricht an den Server. Eine oder mehrere „ausgefüllte“ Instanzen dieses Modells und eine Beschreibung der Benutzeroberfläche. Im Vergleich dazu sind diese drei Teile bei HTML-Formularen in einer Struktur vereint. Eine Unterscheidung zwischen Logik und Aussehen wird bei HTML nicht gemacht. Zum Modell gehören auch Bedingungen und Relationen unter den einzelnen Formularelementen. Dadurch können eine ganze Reihe von Konsistenzprüfung im Browser durchgeführt werden wodurch die Belastung des Servers sinkt und die Performanz der gesamten Anwendung steigt. Jede Komponente des Modells hat einen Typ, der über ein Schema spezifiziert werden kann. Dadurch können selbst eigentlich frei Eingaben über Eingabefelder Regeln unterworfen und vor dem Übertragen überprüft werden. Auch der Nutzer kann darauf hingewiesen werden. Ein Beispiel hierfür ist zum Beispiel ein Eingabefeld für Datumsangaben, die eine entsprechende Form haben müssen. XForms kann in XHTML (die XML konforme Variante von HTML) eingebettet werden, jedoch gibt es auch die Möglichkeit Formulare ohne eine Einbettung darzustellen. Die Steuerelemente werden nach ihrem Zweck beschrieben und nicht nach ihrer Form. D.h. man bestimmt nicht, dass an einer Stelle mehrere Umschaltfelder kommen, sondern dass hier eine Auswahl 1-aus-n getroffen wird. Der Tag würde in diesem fall *select1* heißen und durch seine Kindelemente würde beschrieben, welche Möglichkeiten zur Auswahl stehen. Die Entscheidung ob mehrere Umschaltfelder oder ein Listenauswahlfeld dargestellt werden, trifft der darstellende Browser. Die Auswahl der Steuerelemente wird auf die darstellende Instanz verschoben. Dies hat den Hintergrund, dass XForms auf allen möglichen Geräten dargestellt werden sollen. Von 20" Bildschirmen über PDAs bis hin zu Geräten, die keine Display haben und nur über Lautsprecher kommunizieren. Das heißt, die jeweilige darstellende Applikation muss die Entscheidung treffen, wie Auswahl dem Nutzer, unter Berücksichtigung der vorhandenen Ressourcen, präsentiert wird. Im Beispiel würde das bedeuten, dass auf einem PDA eine Combobox dargestellt werden könnte, da dieses Steuerelement sehr platzsparend ist. Auf einem normalen Computer hätte man die Wahl zwischen Listenauswahlfeld und Umschaltfeldern und bei einem Geräte, dass nur über Sprachausgabe arbeitet, müssten die zur Auswahl stehenden Möglichkeiten vorgelesen werden. Das XForms Formular wäre jeweils das Gleiche ohne jegliche Anpassung an die jeweilige Plattform. In der Spezifikation ist vorgesehen, dass eine darstellende Applikation die Typen der Eingabefelder analysiert und falls vorhanden ein spezialisiertes Steuerelement darstellt. Zum Beispiel könnte die Applikation erkennen, dass ein bestimmtes Eingabefeld ein Datum erwartet. Anstatt nun den Benutzer das Datum in Textform eingeben zu lassen und die Syntax danach zu testen, was unter Umständen mehrere Korrekturen durch den Benutzer notwendig macht, stellt die Applikation einfach ein Kalender-Steuerelement dar. Dies beschleunigt die Auswahl des gewünschten Datums, macht Falscheingaben (die nicht der Syntax entsprechen) unmöglich und als Nebeneffekt tritt das Problem der verschiedenen Formate für Datumsangaben erst gar nicht auf (Tag-Monat-Jahr vs. Monat-Tag-Jahr). Internationalisierung wird auch sonst groß geschrieben, da man in einem XForms Formular die Benutzeroberfläche für mehrere Sprachen spezifizieren kann. Die darstellende Applikation trifft die Entscheidung darüber, welche davon dargestellt wird nach den Spracheinstellungen des Systems auf der sie läuft.

Der XForms Standard enthält noch viele weitere Komponenten, die keine Wünsche offen lassen sollen. Dazu gehört auch die Möglichkeit aus einer vorgegebenen Instanz des Modells die Benutzeroberfläche zu generieren. Dies könnte zum Beispiel bei einem Adressbuch genutzt werden: Das Modell beschreibt ein Adressbuch mit mehreren Einträgen, die jeweils Name, Adresse, Telefonnummer, etc. enthalten. Die Instanz enthält eine solches Adressbuch mit

mehreren Einträgen. Die Benutzeroberfläche kann nun aus der Instanz generiert werden, indem über alle Adressbucheinträge iteriert wird und dabei für jeden Eintrag entsprechende Steuerelemente generiert werden. Dadurch könnte man das gesamte Adressbuch in einem Formular bearbeiten. Auch Bookmarks oder ähnliche Daten können so über ein einziges XForms Formular bearbeitet werden. Beim Übertragen an den Server wird das aktualisierte Adressbuch mitgeschickt.

Der Standard ist ziemlich groß und enthält ferner ein Ereignissystem und ein Aktionssystem um die Formulare interaktiver zu gestalten. Weitere Informationen findet man in [3].

2.3 XAML

XAML stand für eXtensibel Avalon Markup Language, wurde aber in eXtensible Application Markup Language uminterpretiert. „Avalon“[2] ist der Codename des User Interface Systems der nächsten Version von Microsoft's Windows Betriebssystems (Codename „Longhorn“). Auch „XAML“ wird ab und zu als Codenamen gehandelt, jedoch scheint sich dieser Name zu festigen, da er nur von Entwicklern benutzt wird und kaum in der Öffentlichkeit auftaucht. Was XAML inhaltlich bedeutet wird auf jeden Fall erhalten bleiben, da Microsoft die Entwicklung mit viel Geld und vielen Entwicklern voran getrieben und in seine Developer Suit für 2005 fest integriert hat. Bei XAML (ausgesprochen als „Zähml“, mit einem weichen 'Z') handelt es sich um eine Markup-Sprache mit der man Anwendungsoberflächen beschreiben kann. Sie basiert auf XML. Man kann mit XAML die Benutzeroberfläche beschreiben indem die Steuerelemente, die die Benutzeroberfläche ausmachen, angeben werden, zusammen mit ihren Parametern zur Initialisierung. Ebenso werden die Namen der Ereignisbehandlungsroutinen angegeben. Die dazu gehörenden Routinen werden in einer .NET Sprache geschrieben, wie z.B. Visual Basic (VB), C#, (Managed) C++, etc. . Die Programmtexte dafür kann in einer der jeweiligen Quelldateien liegen oder er kann im Falle von C# oder VB auch direkt im XAML Dokument liegen. Um eine mit XAML erstellte Applikation zu erstellen, müssen alle dazu gehörenden Dateien (Quell- und XAML-Dateien) compiliert werden. Der Grund für die Entwicklung von XAML liegt in der Erkenntnis, dass viele Applikation beim Start die Benutzeroberfläche einmal erzeugen und dann auf Ereignisse warten, die die eigentlichen Funktionen der Applikation auslösen. Die Idee ist nun, dass man die Benutzeroberfläche und die Ereignisbehandlungsroutinen in einem XAML-Dokument beschreibt, da sie zusammen gehören, und das die Behandlungroutinen die eigentliche Applikationslogik auslösen, die in reinen Quelldateien geschrieben wird. Es soll noch auf einen Sache hingewiesen werden: der Grund dafür das nur .NET Sprachen aufgeführt sind liegt darin, dass das neue „Avalon“ User Interface System nur ein .NET-API besitzt (nur im Managed Code). Deswegen ist die Programmierung mit Sprachen, die nicht .NET-fähig sind nicht möglich. Das generelle XAML-System wie Objekte konstruiert und initialisiert werden, ließe sich auch auf andere objektorientierte APIs übertragen.

XAML wurde zwar für den oben beschriebenen Zweck entwickelt, stellt im Kern eine 1-zu-1 Beziehung zwischen der .NET Klassenbibliothek und den XML-Tags dar [4]. XAML ist eine Möglichkeit Applikationen zu initialisieren, in dem eine Reihe von Objekten erzeugt und mit bestimmten Werten initialisiert werden. Man kann jedes XAML-Dokument von Hand in einer .NET-Sprache schreiben, jedoch eignet sich XAML sehr gut für hierarchische Initialisierungen wie bei Benutzeroberflächen. Eine wichtige Eigenschaft ist, dass man Parameter von Objekten nicht nur mit Standardtypen wie Int, Bool, String etc. initialisieren kann, sondern auch Objekte

2 Avalon ist ursprünglich der Name der mystischen Insel auf der König Arthur, der Sage nach, begraben liegt

erzeugen und an ein höheres Objekt als Parameter übergeben kann. Zum Beispiel ist ein „Brush“, der die Fläche eines Steuerelements beschreibt, ein Objekt und kann doch sehr leicht für das Steuerelement spezifiziert werden. Zum einen durch praktische Abkürzungen, wie „red“, was vom Compiler automatisch in einen vom System definierten Standard-Brush umgewandelt wird, oder durch die Konstruktion eines Brush-Objektes, bei dem einem alle Möglichkeiten offen stehen. Ein Vorteil dieses Systems, und vor allem der 1-zu-1 Beziehung zwischen Tags und .NET-Klassenbibliothek, ist, dass bei Erweiterungen der „Avalon“ API, diese automatisch mit XAML genutzt werden kann. Das XAML System muss nicht aktualisiert werden.

Man kann XAML eigentlich nicht direkt mit HTML-Forms oder XForms vergleichen, da XAML viel genereller ausgerichtet ist als die andere beiden. Die vorhandenen Steuerelemente sind nicht beschränkt. Sollte ein gewünschtes Steuerelement noch nicht vorhanden sein so kann es durch Elementare Zeichenoperationen erzeugt werden, was durch die zentrale System-API möglich ist. Es gibt keine Limitierung, die die Gestaltung von Formularen oder besser Fenstern einschränkt. Eine weitere nette Fähigkeit von XAML ist, dass es „Styles“ unterstützt, so dass wie bei CSS mehrere Designs definiert werden können, die das Aussehen der Benutzeroberfläche anpassen. Eine mögliche Anwendung sind Personalisierte Anwendung, die sich je nach Nutzer anders Präsentieren: Verspielt für Kinder, cool für Jugendliche und normal für Erwachsene [5].

XAML bietet zwar viel, eine spezielle Infrastruktur für Webformulare ist aber nicht direkt vorhanden. Man kann in XAML nicht einfach eine URL angeben und eine Abschicken-Schaltfläche definieren, die dann automatisch alle eingetragenen Daten an den Server schickt. Man müsste diese Funktionalität selber Programmieren, was dann nicht mehr mit XAML geschehen würde, sondern über die entsprechende .NET-Sprache. Auch Validierung müsste von Hand programmiert werden. Die Applikationsentwicklung wird durch XAML beschleunigt, vor allem wenn viele Fenster und Dialoge gestaltet werden müssen. Bei erscheinen der 2005 Developer Suit von Microsoft sollen bereits visuelle Editoren für XAML bereit stehen, so dass sowohl in Markup als auch per Drag-and-Drop entwickelt werden kann. Microsoft will durch eine „click once“ getaufte Technologie die Grenzen zwischen Webapplikation und normaler Applikation verschwinden lassen. Wie bei Java Webstart soll auf einen Klick eine Applikation herunter geladen, installiert und gestartet werden, so dass spezialisierte Formulare sich einfach auf dem lokalen Rechner ausführen lassen und sich dort als normale Applikationen in einem Browserfenster präsentieren [6].

2.4 VIEWS

VIEWS steht für Vendor Independent Extensible Windowing System. Es handelt sich um ein, auf XML basierendes System zur Gestaltung von Benutzeroberflächen für .NET Anwendungen. Es wurde von an der University of Pretoria, South Africa von Prof. Judith Bishop und Prof. Nigel Horspool von der University of Victoria, Canada entwickelt [7]. Ziel war es eine sehr einfache Beschreibungssprache für Benutzeroberflächen zu entwickeln, um auch in einem Anfängerbuch über C# [8] mit graphischen Benutzeroberflächen zu arbeiten. Das ganze dient zur Abstraktion und Vereinfachung der normalen Oberflächen-API, in diesem Fall Windows.Forms unter .NET. Bis vor kurzem war auch nur die .NET Variante einsatzbereit, so dass das „Vendor Independent“ nicht ganz berechtigt war. Es gibt jedoch einen Rotor Port mit Hilfe von Tcl/Tk, wobei Rotor eine „shared source“ Variante der CLR (Common Language Runtime) ist, also eine .NET Umgebung für nicht Windows Plattformen, jedoch ohne das Windows.Forms API. VIEWS wurde vor kurzem von Karlsruher Studenten auf Mono und Qt

portiert. Mono ist ein echtes Open-Source-Projekt, das die CLR auch unter Unix/Linux zur Verfügung stellt und im Moment die leistungsfähigste .NET-Alternative darstellt.

Die Benutzeroberfläche wird in einem XML-Dokument beschrieben und zur Laufzeit wird nach dieser Beschreibung die Oberfläche mit dem jeweiligen API (Windows.Forms, Tcl/Tk oder Qt) konstruiert, also nicht wie bei XAML kompiliert. Die VIEWS Bibliothek kann von allen .NET-Sprachen benutzt werden. Der Zugang zu den so geschaffenen Steuerelementen ist auf das Nötigste beschränkt, ganz nach der Maxime der Einsteigerfreundlichkeit. Von der Mächtigkeit kann man das VIEWS-System als Fliege im Vergleich zum Elefanten XAML sehen. Die Anzahl der zur Verfügung stehenden Steuerelemente ist beschränkt auf diejenigen, die schon Implementiert wurden, dies Umfasst: Schaltflächen, statischen Text, Eingebefelder, Auswahlfelder, Umschaltfelder, Mehrfachlistenauswahlfelder, Auswahlfelder, Bildsteuerelemente, Schieberegler, Fortschrittsanzeigen und Öffnen-/Speicherndialoge. Im Vergleich zu HTML-Formularen fehlen mehrzeilige Eingabefelder, Listenauswahlfelder und Eingabefelder im Passwortmodus. Es wäre sicher möglich weitere Steuerelemente zu implementieren offenbar scheinen die Gegebenen für die Beispiele des Buchs auszureichen; VIEWS richtet sich nicht an „echte“ Entwickler. Ein Nachteil im Vergleich zum XAML-System ist, das für weitere Steuerelemente immer die Bibliothek erweitert werden muss. Eine Automation wie bei XAML ist nicht vorhanden, was auch nicht wundert da es nicht leicht ist den gleichen Wrapper für drei verschiedene Systeme zu erzeugen.

2.5 PDF

Das PDF Format, der Firma Adobe ist seit Jahren ein fester Standard für hochauflösende oder komplizierte Dokumente. Oft wird es dort eingesetzt wo die Layouteigenschaften von HTML nicht mehr ausreichen. Die Popularität des PDF-Formats kommt unter anderem daher, dass die Betrachtungsapplikation, Acrobat Viewer, kostenlos erhältlich ist. Es gibt zwar auch andere Applikationen die PDF-Dateien darstellen und drucken können, jedoch sind diese oft den neusten Versionen von PDF hinterher. Adobe hat zwar das Format offen gelegt und auch gut Dokumentiert, treibt die Entwicklung jedoch alleine voran.

PDF-Dateien sind oft das Ende einer Dokument-Pipline, da es kaum Anwendungen gibt, die PDF-Dateien weiterverarbeiten. Meistens wird das PDF-Format als Container für ein fertiges Dokument genutzt und an den Endbenutzer zum Betrachten oder Drucken ausgeliefert [9]. Die Stärken des PDF-Formats besteht in der Möglichkeit beliebige Grafik (sowohl vektor- als auch pixelbasierte) Geräteunabhängig zu speichern und auf jedem System darzustellen. Zusätzlich wurden weitere Fähigkeiten entwickelt, die zum Teil nur von den Adobe Produkten genutzt werden. Zu deren Fähigkeiten gehören Anotationen, Sounds, Videos und auch Formulare. Manche Erzeuger des PDF-Formats nutzen auch die Möglichkeit, die Dateien zu verschlüsseln, zu signieren oder mit Limitierungen zu versehen (es darf nicht kopiert werden, etc.).

Das PDF-Format ist nur zum Teil textbasiert. Ein PDF Dokument besteht aus einer Sammlung von Objekten, die das Dokument beschreiben. Diese werden durch eine recht einfache textuelle Syntax beschrieben. Manche dieser Objekte enthalten sog. Streams, in denen die Zeichenanweisungen in einer PostScript ähnlichen Form gespeichert sind. Meistens werden diese Ströme per Deflate-Algorithmus komprimiert, so das sie nicht mehr direkt lesbar sind. Das PDF-Format ist ein gewachsenes Format, so dass die Struktur der einzelnen Objekte sehr kompliziert geworden ist und ebenso das Zusammenspiel der verschiedenen Objekte, was das

„Wiedereinlesen“ von PDF-Dokumenten nicht leicht macht [10].

Das in PDF vorhandene Formularsystem stellt alle Steuerelemente zur Verfügung, die es auch bei HTML-Formularen gibt. Zusätzlich bietet es auch ein RichText-Steuerelement an um primitiv gelayouteten Text einzulesen. Die GET und POST Sendemethoden an einen beliebigen Server sind vorhanden. Das System versucht in keiner Weise innovativ zu sein, sondern die gleichen Möglichkeiten wie HTML-Formulare anzubieten. Dieses Formularsystem wird jedoch in naher Zukunft von einem neuen System ersetzt. Dieses von Adobe entwickelte System trägt den Namen XFA (XML Forms Architecture)[11] und basiert, wie der Name schon andeutet, auf XML. Es gehört zur neuen Adobe XML Architecture und ist Indiz dafür das Adobe komplett auf den XML-Zug aufspringen will. Auch das eigentliche PDF-Format soll in einen XML Container eingebettet werden, wobei bis jetzt XFA-Formulare als Strom innerhalb eines PDF-Objektes gespeichert wurden und nicht über reines XML zu erreichen sind. Jedoch ist das XFA-System noch nicht voll integriert und hat sich noch nicht bewährt, deshalb wurde vor allem das bisherige System betrachtet und auch implementiert [12].

2.6 AWT

Das Abstract Window Toolkit ist ein Teil der Java Foundation Classes und stellt ein API zur Gestaltung von graphischen Benutzeroberflächen dar. Der Grund warum AWT hier betrachtet wird ist der, dass der in der Implementierung verwendete FO-Prozesser, FOP, auch einen AWT-Renderer besitzt, d.h. direkt FO-Dokumente darstellen kann. Es gab die Überlegung auch den AWT-Renderer um Formulare zu erweitern.[3] Das AWT stellt eine Ansammlung von Klassen dar, mit deren Hilfe in Java Plattform unabhängig Benutzeroberflächen gestaltet werden können. Dabei bilden die Klassen eine Abstraktion, der auf allen graphischen Benutzeroberflächen vorhandenen Steuerelemente. Es handelt sich aber um einen kleinsten Nenner, der Steuerelemente, jedoch gibt es auf jedem System elementare Zeichenoperationen, so dass durch eigene Zeichenroutinen fast jedes Steuerelement erzeugt werden kann. Ein wesentliches Problem besteht darin, dass diese von Hand gezeichneten Steuerelemente nicht den Standards von allen Systemen entsprechen und einen inkonsistenten Eindruck hinterlassen. Jedoch kann so vollkommen Plattformunabhängig entwickelt werden. Fast genau so wie bei XAML gibt es keine Einschränkung der vorhandenen Steuerelemente, da man Zugriff auf eine ganzes System-API nutzt, jedoch handelt es bei AWT um eine Klassenbibliothek und nicht um eine Markup-Sprache durch die Oberflächen beschrieben werden. Das Erzeugen der Steuerelemente geschieht über Java-Code. Eine Formularinfrastruktur zum Sammeln und Versenden von Daten ist ebenfalls nicht vorhanden, lässt sich jedoch genau so wie bei .NET mit Hilfe der Klassenbibliothek realisieren [13].

2.7 Gemeinsamkeiten

Bis auf VIEWS, erreichen alle Systeme, ob zur Darstellung oder zur Beschreibung von Formularen, die HTML-Formular "Maßstäbe". In diesem Kern an Steuerelementen gibt es kaum Unterschiede und es herrscht Einigkeit, dass auch jedes einzelne Steuerelement nötig ist. Zum Vergleich hier die unterstützten Steuerelemente der verschiedenen Systeme, beschränkt auf diejenigen die für Formulare relevant sind.

3 Gründe warum die Überlegung nicht umgesetzt wurde finden sich in Abschnitt 4.5

System	HTML Forms	XForms	XAML	VIEWS	PDF	AWT
Schaltfläche	Ja	Ja	Ja	Ja	Ja	Ja
Eingabefeld	Ja	Ja	Ja	Ja	Ja	Ja
Mehrzeil. Eingabefeld	Ja	Ja	Ja	Nein	Ja	Ja
Umschaltfeld	Ja	Ja	Ja	Ja	Ja	Ja
Auswahlfeld	Ja	Ja	Ja	Ja	Ja	Ja
Listenauswahlfeld	Ja	Ja	Ja	Nein	Ja	Ja
Mehrf. Listenauswahlf.	Ja	Ja	Ja	Ja	Ja	Ja
Passwort	Ja	Ja	Ja	Nein	Ja	Ja
Öffnen-/Speicherndialog	Ja (Upload)	Ja (Upload)	Ja	Ja	Ja	Ja
Schieberegler[4]	Nein	Ja	Ja	Ja	Nein	Ja
Richedit	Nein	Nein	Ja	Nein	Ja	Ja
Tooltips[5]	Ja	Ja	Ja	Nein	Ja	Ja

3 Die Erweiterung

Ein wichtiger Faktor für die Entwicklung der Erweiterung war Zeit. Als Rahmen wurden 3 Monate angesetzt, in denen die Analyse, der im vorherigen Kapitel beschriebenen Systeme, die Entwicklung der Erweiterung und die Implementation statt finden sollte. Deshalb wurde versucht, die angestrebten Eigenschaften der Erweiterung dem Zeitrahmen anzupassen, so dass auch eine vollständige Implementierung entwickelt werden konnte.

3.1 Einflüsse

Ein zentraler Einfluss war die Trennung von Modell, Instanz und Beschreibung der Steuerelemente des XForms-Systems. Es werden jedoch Modell und Instanz als eine Einheit betrachtet, weil die Formulare nicht mehrere ausgefüllte Instanzen des Modells bearbeiten können sollen. Jedes Formular, es können mehrere pro Dokument sein, besitzt ein Modell das beschreibt, welche Elemente zum Formular gehören. Gleichzeitig werden auch schon Werte angegeben. Diese können als Vorgabe, Beispiel oder als bereits ausgefüllte Werte dienen. Werden diese Werte nicht vorgegeben, so wird das Formular als noch nicht ausgefüllt dargestellt. Davon getrennt werden innerhalb des Dokuments und der einzelnen Seiten die Steuerelemente[6] beschrieben, die die Eingabemöglichkeiten darstellen. Alle Eigenschaften der Steuerelemente die ihr Aussehen oder ihr Layout betreffen, werden bei den Steuerelementen im Dokument spezifiziert. Die Komponenten des Formulars oder der Nachricht an den Server werden im Logikteil mit Angaben über ihren Typ oder vorgegebenen Werten beschrieben.

Die Auswahl der verschiedenen Typen die im Logikteil verwendet werden können, ergibt sich aus der Analyse von Formularen (siehe Kapitel 1.2) und der Beschreibung der Eingabemöglichkeiten bei XForms (nicht des Modells). Es gibt die einzeilige freie Eingabe für kurze textuelle oder textähnliche Angaben (*edit*); die mehrzeilige textuelle Eingabe für Kommentare oder längeren Text (*multiline*); die Auswahl einer Möglichkeit aus vielen (1-aus-n,

4 Das Schieberegler Steuerelement ist eine Möglichkeit numerische Daten in einem bestimmten Bereich abzufragen, es ist sehr praktisch wenn man eine Anzahl in Erfahrung bringen will. Dies muss dann nicht über ein Eingabefeld geschehen, wo auch Nichtzahlen eingetippt werden können.

5 Tooltips sind keine echte Steuerelemente, sie gehören meistens zu einem Steuerelement und stellen einen Hinweis oder weitere Informationen dar, durch sie können Formulare benutzerfreundlicher gestaltet werden.

6 Steuerelemente sind in Bezug auf papierbasierte Formulare als die sinngemäß entsprechenden Kästchen bzw. Freiräume zu sehen

select1) und die Auswahl von mehreren Elementen aus vielen (k-aus-n, *selectn*). Dadurch lassen sich alle Angaben von einem Nutzer erfassen. Zudem wurde noch ein versteckter Typ (*hidden*), nach dem Vorbild von HTML-Formularen hinzugenommen, der nicht durch ein Steuerelement dargestellt wird und damit nicht vom Nutzer verändert werden kann, er existiert also nur im Logikteil. Dort werden auch schon die Funktionen von Schaltflächen (*button*) festgelegt. Sie stellen keine Datenelemente dar, sondern definieren Versand- und Rücksetzaktionen. Sie werden nicht erst im Layoutteil deklariert, da eventuell Buttons auch Validierungsaktionen auslösen können und diese dann im Logikteil definiert würden.

Die Auswahl der Steuerelemente, die im Dokument verwendet werden können, richtet sich an HTML-Formularen aus. Es werden alle Steuerlemente außer dem Dateidialog integriert. An dieser Stelle wurde auch die Entscheidung getroffen sich stärker an digitalen Formularen zu orientieren als an Papier basierten. XSL-FO ist zwar auf Papierlayout ausgerichtet, die Hauptverwendung ist im Moment aber die Generierung von PDF Dokumenten, also digitalen Dokumenten. Die Steuerelemente Listenauswahlfeld und Mehrfachlistenauswahlfeld können nicht direkt auf Papier umgesetzt werden, werden jedoch vom Gestalter von digitalen Formularen erwartet. Für Papier basierte Formulare dürfen diese Steuerelement eben nicht verwendet werden oder es werden diese mit Hilfe einer XSLT-Transformation in darstellbare Steuerelemente umgewandelt (siehe 4.6). Zudem gibt es bei digitalen Formularen die Absenden- und Rücksetzschaltflächen, die auf Papier wenig Sinn machen.

3.2 Logikteil

Ein FO-Dokument hat immer die folgende Struktur:

```
<?xml version="1.0" encoding="UTF-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">

    </fo:leader><fo:layout-master-set>
        <fo:simple-page-master master-name="A4" ...>
            ...
        </fo:simple-page-master>
    </fo:layout-master-set>

    <fo:page-sequence master-reference="A4">
        <fo:flow flow-name="xsl-region-body">
            ...
        </fo:flow>
    </fo:page-sequence>

</fo:root>
```

Zuerst kommt die XML Deklaration mit Version und Kodierung (wie bei jedem XML-Dokument), danach kommt das *fo:root* Element in dem auch der fo-Namensraum deklariert wird. Das *fo:root* Element hat ein *fo:layout-master-set* als Kind. Hierin werden in der Form von *fo:simple-page-master* Elementen die verschiedenen Seitentypen definiert, mit Höhe, Breite und den Einzügen auf allen vier Seiten eines Plattes. Eine solcher Seitentyp bekommt einen Namen (hier „A4“) über den er später benutzt wird. Nach dem alle Seitentypen definiert wurden kommt der eigentliche Inhalt des Dokuments in Form von *fo:page-sequence*. Jedes *fo:page-sequence* Element stellt den Inhalt von aufeinander folgenden Seiten des selben Typs dar. Eine Seite enthält fünf Regionen (oben, unten, links, rechts, mitte) wobei "xsl-region-body" ein Bezeichner für den mittleren Hauptteil einer Seite ist (wird im *fo:simple-page-master* festgelegt). In diesen wird dann durch das fo:flow Element der Inhalt der Seite „gegossen“.

Damit die Elemente der Erweiterung beim Einlesen von den FO-Elementen unterschieden werden können, bekommen sie einen eigenen Namensraum (http://www.fzi.de/2004/XSL/FormExtension); als Kürzel wurde *fex* gewähl. Die logische Struktur des oder der Formulare ist global für das ganze Dokument gültig, ähnlich der Definition der Seitentypen. Deswegen werden alle Formulare unter einem *fex:forms* Element zusammen gefasst, das als Kind des *fo:root* Elements nur einmal vorkommt. Hierin werden die Logikteile aller Formulare des Dokuments beschrieben. Ein solches Formular-Element heißt *fex:form*, und in das obige Dokument eingebettet sieht das ganze so aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format"
         xmlns:fex="http://www.fzi.de/2004/XSL/FormExtension">

    <fo:layout-master-set>
        <fo:simple-page-master master-name="A4" ...>
            ...
        </fo:simple-page-master>
    </fo:layout-master-set>

    <fex:forms>
        <fex:form id=“from1“ ...>
            ...
        </fex:form>

        <fex:form id=“from2“ ...>
            ...
        </fex:form>
    </fex:forms>

    <fo:page-sequence master-reference="A4">
        <fo:flow flow-name="xsl-region-body">
            ...
        </fo:flow>
    </fo:page-sequence>

</fo:root>
```

Wie man an dem Beispiel bereits sieht, besitzt jedes Formular einen eindeutigen Bezeichner (*id*) der unter den Formularen nur einmal vorkommen darf. Zudem besitzen Formulare noch zwei weitere Attribute: target und method. Target gibt die URL der Applikation an, die die Daten beim Absenden empfangen soll. Method gibt die Methode an mit der die Daten versand werden sollen. Bis jetzt sind die Methoden POST und GET definiert, denen der Versand über die jeweilige HTTP-Methode entspricht. Weitere Methoden können hier integriert werden, wenn das Zielsystem, auf das gerendert wird, weitere Methoden unterstützt.

In einem *fex:form* Element werden die Komponenten deklariert, die zu diesem Formular gehören. Entsprechend 3.1 gibt es folgende Elemente: *decl-edit*, *decl-multiline*, *decl-select1*, *decl-selectn*, *decl-hidden* und *decl-button*. Alle Elemente haben ein *id*-Attribut um sie zu identifizieren. Dieses wird auch als Schlüssel an den Server geschickt, zusammen mit dem Wert, der dem Steuerelement entnommen wird. Es wurden im Beispiel als Bezeichner immer eine Kombination von Formular-ID und einem speziellen Bezeichner gewählt, damit bei mehreren Formularen in einem Dokument Verwechslungen vermieden werden. Z.B. kann man den Send-Button des Formulars mit dem Namen „form1“ den Namen „form1.sendbutton“ geben, damit er mit anderen Send-Buttons nicht verwechselt wird. Es handelt sich jedoch nur um eine Empfehlung, jeder eindeutige Bezeichner auf Dokumentebene ist zulässig.

Jedes der Deklarationselemente besitzt noch spezielle Attribute, die auf den jeweiligen Zweck ausgerichtet sind.

3.2.1 decl-edit

Das *decl-edit* besitzt neben dem *id*-Attribut noch die Attribute *value*, *default-value* und *regexp*. *Value* ist die Zeichenkette, die in das dazu gehörige Eingabefeld bereits eingetragen ist. *Default-value* ist die Zeichenkette auf die beim Drücken der Rücksetztschaltfläche zurückgesetzt wird. In manchen Situationen ist es vorteilhaft diese beiden Werte zu trennen, zum Beispiel wenn man eine exemplarische Mustermann-Version vorgeben will, aber die Rücksetzschaltfläche alle Werte auf leer bzw. nicht ausgewählt setzen soll. Das *regexp* Attribut stellt einen regulären Ausdruck dar, dem der Inhalt des Editfelds genügen muss. Als Syntax für die regulären Ausrücke wurde die Syntax des java.util.regex APIs gewählt, die der Perl Syntax für Musterabgleich (pattern matching) sehr ähnlich ist. Zu einer Implementierung kam es jedoch nicht; Gründe dafür finden sich in 4.5. Es würde sich um eine eingeschränkte Form der Validierung für freie Zeichenketten handeln, die in einem Eingabefeld eingegeben werden können. Es wäre kein Ersatz für ein Typsystem wie bei XForms, bei dem die Eingabe mit Hilfe von Schema genau beschrieben werden kann. Eine Entwicklung eines Typsystems oder der Einbindung von Schema hätte aber den Rahmen dieses Projekts gesprengt, deshalb wurde das *regexp* Attribut hinzugefügt um anzudeuten wo Validierungsinformationen in der Spezifikation untergebracht werden können.

3.2.2 decl-multiline

Bei *decl-multiline* gibt es keine weiteren Attribute außer dem *id*-Attribut. Die *value* und *default-value* Werte werden über entsprechende Kinder mit Text angegeben, da diese Werte mehrere Zeilen und eventuelle Umbrüche enthalten. Ein *regexp* wird hier nicht benötigt, da bei mehrzeiligen Eingabefeldern nur freier Text eingegeben wird, der keinen speziellen Regeln entspricht.

3.2.3 decl-select1

Das *decl-select1* Element besitzt wieder ein *value* und ein *default-value* Attribut. Die jeweils die eine ausgewählte Möglichkeit angeben. Die Zeichenkette in *value* und *default-value* ist ein Bezeichner für eine Möglichkeit, diese Zeichenkette wird auch als Wert zum Schlüssel *id* des *decl-select1* übermittelt. Die zur Verfügung stehenden Möglichkeiten werden erst durch zum *decl-select1* gehörenden Steuerelemente bestimmt. Hierfür gibt es zwei verschiedene: Eine Gruppe von Umschaltfeldern oder ein Listenauswahlfeld. Bei den Umschaltfeldern bekommt jedes Umschaltfeld den entsprechenden Wert zugewiesen dem er entspricht. Bei einem Listenauswahlfeld werden die Möglichkeiten die angeboten werden bei dem Listenauswahlfeld spezifiziert (siehe 3.3.4). Eine Angabe der Auswahlmöglichkeiten im Logikteil wäre zwar eine konsequentere Trennung von Logik und Erscheinung, jedoch müssen die Möglichkeiten auch bei den Steuerelementen angegeben werden, so dass eine Trennung nur über eine weiter Indirektionsebene möglich wäre, indem neue Bezeichner für die Auswahlmöglichkeiten vergeben werden. Man könnte auch die Auswahlmöglichkeiten doppelt angeben, jedoch führt dies leicht zu Inkonsistenzen, wenn der eine Teil verändert wird und diese Veränderungen im anderen Teil vergessen werden.

3.2.4 decl-selectn

Sehr ähnlich sieht auch das *decl-selectn* Element aus. Der einzige Unterschied besteht darin, das

ein *decl-selectn* mehrere Werte gleichzeitig haben kann. Deshalb müssen auch in *value* und *default-value* mehrere Werte angegeben werden können, dies geschieht über Komma getrennte Listen von Bezeichnern. Beim Versand an den Server wird der Schlüssel mehrmals übertragen, jeweils mit einem anderen ausgewählten Wert. Dies wird auch bei HTML-Formularen so gemacht, da es dem HTTP-Protokoll entspricht.

3.2.5 decl-hidden und decl-button

Das *decl-hidden* Element besitzt nur ein *value* Attribut, zusätzlich zum *id*-Attribut. Ein Rücksetzen ist nicht notwendig, da es nicht verändert werden kann. Die Deklaration einer Schaltfläche geschieht über ein *decl-button* Element. Neben dem *id*-Attribut besitzt es noch ein *action* Attribut, dass angibt was die zugehörige Schaltfläche auslöst. Bis jetzt sind *SUBMIT* und *RESET* definiert, die die Standardaktionen auslösen. An dieser Stelle könnten aber auch Funktionen einer Skriptsprache, wie ECMA-Script angegeben werden. Es muss dies jedoch vom Zielsystem angeboten werden.

Um das ganze zu veranschaulichen sei hier ein beispielhafter Logikteil angegeben:

```
...
<fex:forms>
  <fex:form id="form1" target="www.fzi.de/test.php" method="GET">
    <fex:decl-hidden id=“form1.customer“ value=“id12345“/>
    <fex:decl-edit id="form1.name" value="Musterman" default-value=""/>
    <fex:decl-edit id="form1.fname" value="Max" default-value=""/>
    <fex:decl-selectn id="form1.icecream" value="mint, chocolate"/>
    <fex:decl-select1 id="form1.topping" value="chocolate"/>
    <fex:decl-selectn id="form1.special" value="Straw, Umbrella"/>
    <fex:decl-multiline id="form1.comment">
      <fex:value>Anything else to say?</fex:value>
      <fex:default-value>Comment?</fex:default-value>
    </fex:decl-multiline>
    <fex:decl-select1 id="form1.payment" value="Mastercard"/>
    <fex:decl-button id="form1.sendbutton" action="SUBMIT"/>
    <fex:decl-button id="form1.resetbutton" action="RESET"/>
  </fex:form>
</fex:forms>
...
```

3.3 Layoutteil

Beim Logikteil ist die Interaktion mit der FO-Struktur gering, beim Layout aber umso größer. Bei der Einbettung von neuen Komponenten in die Beschreibung der Blöcke und des Textes muss die Struktur des Dokuments gewahrt werden. Eine Möglichkeit wäre eine zweite Ebene über dem Textlayout zu schaffen, in der nur Formularelemente und deren Position sowie Aussehen definiert werden. Im Textlayout müssten dann an den Stellen der Formularelemente Lücken geschaffen werden. Der Nachteil dieses Ansatzes ist, dass das Layout der Steuerelemente extra angegeben werden muss und bei jedem ändern des Formulars müssen sowohl die Textebene als auch die Formularebene angepasst werden. Die Steuerelemente wären also nicht wirklich in einem FO-Dokument integriert. Um die Integration zu verbessern wurden Standardkomponenten eines FO-Layouts genommen und spezielle Formularversionen davon konstruiert. Zwei Komponenten kamen hierfür in Frage: zum einen das *fo:block* Element und zum anderen das *fo:inline* Element.

Das *fo:block* Element dient dazu die Seite in rechteckige Blöcke zu unterteilen, wobei zwei

Blöcke nicht neben einander, sondern immer untereinander liegen. In einem Block können weitere Unterblöcke definiert werden. Bei jedem Block kann bestimmt werden wie der Rahmen aussieht und wie groß die Einzüge auf allen Seiten sind, ebenso Angaben, die die Schriftart betreffen. Auf einer Seite werden meistens zuerst Blöcke definiert, die dann mit Text gefüllt werden. Ein Paragraph wäre zum Beispiel ein solcher Block. Im Textfluß selbst können mit *fo:inline* Elementen Veränderungen der Schrift oder der Farbe vorgenommen werden. Da Formularelemente normalerweise rechteckig sind, lagen die *fo:block* Elemente sehr nahe, jedoch ist es schwer Blöcke in fließenden Text zu integrieren, was vor allem für Lückentexte ein Nachteil wäre. Da die Beschriftung von Formularelementen über FO geschehen soll, wurde zugunsten der starken Integrierung in fließenden Text entschieden und damit wurde das *fo:inline* Element als Basis für die Formularelemente gewählt. D.h. ein Formularelement soll so einfach wie ein *fo:inline* Element im Text untergebracht werden. Ein weiterer Vorteil ist das durch die Integration in Text viele Parameter für das Aussehen der Steuerelemente bereits bestimmt ist. Die Höhe eines Eingabefelds ist zum Beispiel durch die Zeilenhöhe bestimmt, die Schriftart und Größe durch den umgebenden Text. Dadurch schrumpft die Anzahl der Attribute, die nötig sind um ein Eingabefeld oder ein anderes Steuerelement zu bestimmen.

Alle Steuerelemente haben einige Attribute gemeinsam, diese wären *ref*, *name*, *hint* und *taborder*. *Ref* gibt das *decl*-Element an dem dieses Steuerelement zugeordnet ist. Hier muss der Typ übereinstimmen, d.h. der *ref* Wert eines Eingabefelds muss mit dem *id*-Attribut eines *decl-edit* übereinstimmen, der eines Umschaltfelds mit einem *decl-select1*, usw.. *Name* gibt dem Steuerelement eine Namen, der bei Fehlermeldungen oder Hinweisen verwendet wird, er muss deshalb für den Nutzer verständlich sein. *Hint* gibt einen Hilfetext an, der als Tooltip dargestellt werden kann. *Taborder* gibt, wie der Name schon andeutet, einen Rang in der Reihenfolge aller Steuerelemente an, die beim drücken der Tab-Taste durchlaufen wird. Vor allem bei großen Formularen kann so das Ausfüllen beschleunigt werden, da zum Wechseln des Steuerelementes nicht zur Maus gegriffen werden muss. Dies nutzt Anwendern am meisten, die viele Daten über solche Formulare eingeben müssen. Jedes Element besitzt darüber hinaus spezifische Attribute, die nötig sind um das jeweilige Steuerelement vollständig zu beschreiben.

3.3.1 edit

Ein Eingabefeld wird über ein *edit* Element definiert. Als zusätzliche Attribute besitzt es ein *password* Flag, das angibt ob das Steuerelement die Eingabe durch Punkte bzw. Sternchen ersetzten soll (Standardwert ist false), und ein *width* Attribut, das die Breite des Eingabefeldes bestimmt. Die Breite kann wie andere Längenangabe in FO in verschiedenen Maßeinheiten angegeben werden, wie „30mm“ oder „40pt“. Man hätte die Breite eines Eingabefelds auch in Zeichen angeben können, jedoch ist durch die genau Länge das Ergebnis leichter vorherzusehen, da nicht erst über die aktuelle Schriftart und Größe die endgültige Breite errechnet werden muss. Die Eingabe im Eingabefeld ist durch horizontales Scrollen auch nicht auf die vorgegebene Breite beschränkt.

3.3.2 multiline

Das Kommentarfeld (mehrzeiliges Eingabefeld) wird über ein *multiline* Element definiert. Es besitzt als weitere Attribute einen *width* und *height* Wert, die die Dimensionen des Steuerelements bestimmen. Dabei wir die Zeile in der das Feld liegt auf die Höhe height vergrößert, sofern diese größer ist. Die Breite funktioniert genau so wie beim *edit* Element. Die Angaben bei *width* und *height* können wiederum in verschiedenen Maßeinheiten angegeben werden. Beim vergrößern einer Zeile wird der Text an der oberen Kante der Zeile ausgerichtet.

Eventuell ist in Zukunft hier eine Auswahl nötig, so dass über ein text-align Attribut die Ausrichtung des umgebenden Textes oder anders herum die Ausrichtung des Kommentarfeldes bestimmt werden könnte.

3.3.3 radiobutton und checkbox

Umschaltfelder (*radiobutton*-Element) und Auswahlfelder (*checkbox*-Element) sind recht ähnlich. Sie besitzen nur ein weiteres *value* Attribut, das bestimmt welche mögliche Auswahl sie darstellen. Alle Umschaltfelder bzw. Auswahlfelder, die den selben *ref* Wert haben, formen eine Gruppe. Die Umschaltfelder sind dabei einem *decl-select1*-Element zugeordnet und es ist höchstens eines davon markiert. Die Auswahlfelder sind einem *decl-selectn* zugeordnet. Es können beliebig viele von ihnen markiert sein. Zu beginn sind diejenigen markiert deren *value* in der Komma getrennten *value*-Liste des *decl-selectn* vorkommen. Bei Umschaltfeldern ist dies maximal einer. Die Rücksetzwerte werden entsprechend festgelegt.

3.3.4 combobox

Als Alternative für eine Gruppe von Umschaltfeldern gibt es das Listenauswahlfeld. Dieses wird über ein *combobox*-Element definiert. Als einziges weiteres Attribut gibt es *width*, das die Breite des Listenauswahlfeldes angibt genau so wie bei Eingabefeldern. Die Höhe leitet sich wieder von der Höhe der Zeile ab. Die Auswahlmöglichkeiten des Listenauswahlfeldes werden durch die Kinder des *combobox* Elements bestimmt. Dort wird für jede Möglichkeit ein *option*-Element angegeben, das die Attribute *name* und *value* besitzt. *Name* ist der Text der in dem Listenauswahlfeld dargestellt wird und *value* bezeichnet die Auswahlmöglichkeit, der der Text entspricht. Über *value* wird auch die Vorauswahl über das *value* Attribut des zugehörigen *decl-select1*-Elements vorgenommen.

3.3.5 listbox

Analog zum Listenauswahlfeld gibt es ein Merhfachlistenauswahlfeld, mit entsprechendem *listbox*-Element. Zum *width* Attribut gibt es noch ein *height* Attribut, das die Höhe des Steuerelements wie bei einem Kommentarfeld angibt. Die Auswahlmöglichkeiten werden wie bei dem Listenauswahlfeld über die *option* Kinder beschrieben, der einzige Unterschied besteht darin, dass mehrere Möglichkeiten ausgewählt werden können und dieses Steuerelement deshalb einem *decl-selectn* zugeordnet ist.

3.3.6 button

Das *button* Element beschreibt das Aussehen einer Schaltfläche. Es besitzt ein *width* Attribut und übernimmt die Höhe von der Zeile in der es liegt. Es besitzt zudem ein *text*-Attribut, das den Text angibt der innerhalb der Schaltfläche dargestellt wird. Die Aktion die beim Klicken der Schaltfläche ausgelöst wird ist über das zugehörige *decl-button* Element bestimmt.

Um die Einbettung der Steuerelemente in eine FO-Dokument zu verdeutlichen, sei hier ein Ausschnitt eines Formulars gegeben. Die Steuerelemente die nicht vorkommen werden analog zu denen im Beispiel eingebunden.

```
...
<fo:block> Name:
<fex:edit ref="form1.name" hint="family name" width="50mm"/>
Firstname:
```

```
<fex:edit ref="form1.firstname" hint="your first name" width="50mm" />
</fo:block>
<fo:block> Choose the different icecream flavours you want
  <fo:block>
    <fex:checkbox ref="form1.icecream" value="vanilla"/> Vanilla
    <fex:checkbox ref="form1.icecream" value="chocolate"/> Chocolate
    <fex:checkbox ref="form1.icecream" value="lemon"/> Lemon
    <fex:checkbox ref="form1.icecream" value="walnut"/> Walnut
    <fex:checkbox ref="form1.icecream" value="mint"/> Mint
  </fo:block>
</fo:block>
<fo:block> Method of Payment:
  <fex:combobox ref="form1.payment" hint="Payment Method" width="30mm">
    <fex:option name="Visa" value="Visa"/>
    <fex:option name="Mastercard" value="Mastercard"/>
    <fex:option name="American Express" value="American Express"/>
    <fex:option name="Diner's Club" value="Diner's Club"/>
  </fex:combobox>
</fo:block>
...
```

4 Implementierung

4.1 Der FO-Prozessor

Einen eigenen FO-Prozessor zu entwickeln kam nicht in Frage, da dies allein den Rahmen einer Studienarbeit gesprengt hätte. Um einen bestehenden zu erweitern muss man natürlich den Quellcode haben. Die meisten kommerziellen FO-Prozessoren sind nicht quelloffen. Deshalb wurde der quelloffene FO-Prozessor FOP[14] von apache.org benutzt. FOP ist auch recht verbreitet und liegt im Moment in der Version 0.20.5 vor. Er ist noch nicht 100% Standard konform bezüglich XSL-FO 1.0, was aber auch kein anderer FO-Prozessor im Moment (Oktober 2004) ist. FOP ist komplett in Java geschrieben und besitzt viele Renderer für verschiedene Formate, unter anderem PDF, PCL, PostScript, RTF, SVG, XML, Print, AWT, MIF und TXT. Nicht alle Renderer haben den selben Funktionsumfang, zumal mache Formate kein Layout unterstützen und nicht jeder Renderer mit gleich viel Nachdruck entwickelt wurde. Der wichtigste und auch am weitesten entwickelte Renderer ist der PDF-Renderer, der sogar Verschlüsselung und Signierung beherrscht. Eine weitere Eigenschaft die FOP attraktive machte, war das Erweiterungssystem das bereits integriert war. Über dieses System haben die FOP Entwickler bereits eine Erweiterung geschrieben, mit der man das elektronische Inhaltsverzeichnis in PDF-Dateien (PDF Bookmarks) angeben kann. Ebenso eine Erweiterung mit der spezielle Über-/Unterschriften für Tabellen angegeben werden können, falls die Tabelle auf mehrere Seiten aufgeteilt wird (Table Continuation Label). Durch diese schon vorhandenen Erweiterungen erschien auch eine Formularerweiterung einfacher.

4.2 FO-Verarbeitung

Um die Implementierungsschritte, die für eine Formularerweiterung nötig waren, zu erläutern ist ein Verständnis des Ablaufs der Verarbeitung von FO-Dokumenten nötig. Der Prozess nimmt eine FO-Datei als Eingabe und generiert daraus eine Datei im Ausgabeformat. Dabei werden 4 Phasen durchlaufen: Parsing, Refinement, Layout und Rendering. Als erstes wird das FO-Dokument eingelesen (Parsing) und in einem FO-Baum im Speicher abgelegt. Dann wird diese Baumstruktur anhand der Attribute der Knoten des Baums modifiziert (refined FO-Tree). Aus diesem verfeinerten FO-Baum wird dann der Layout-Baum (Area-Tree) generiert, dessen Knoten den Flächen und Texten des Dokuments entsprechen. Bei der Transformation geschieht schon ein Großteil des Layoutprozesses. Manche Layoutentscheidungen werden iterativ bestimmt, vor allem dann wenn mehrere Anforderungen in Konflikt stehen. Aus dem fertigen

Layout-Baum generiert der Renderer dann das Ausgabedokument, in dem er die Flächen und Texte in das entsprechende Format transformiert.

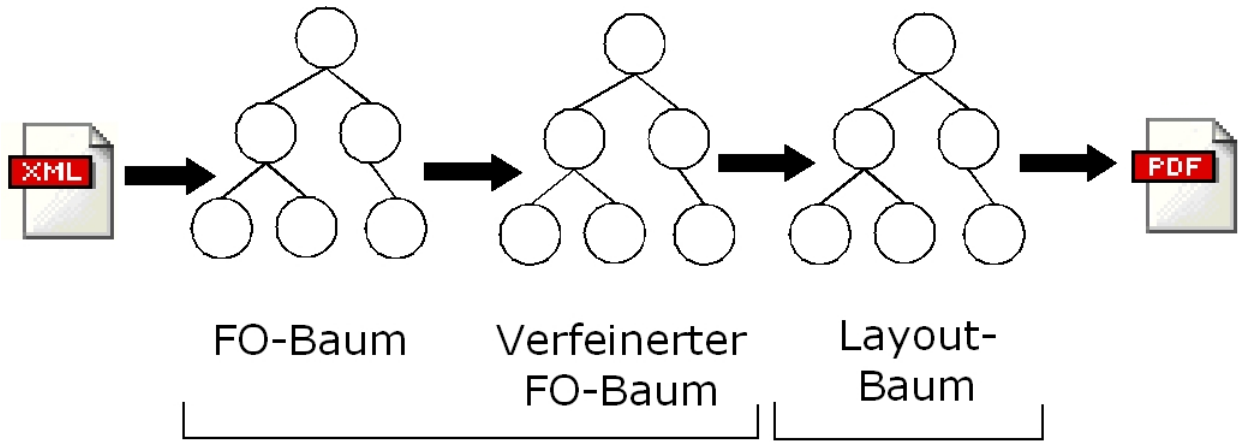

Fig. 3. Ein FO-Dokument wird beim einlesen in einen FO-Baum umgewandelt. Dieser wird dann in einen verfeinerten FO-Baum umgewandelt. Daraus wird ein Layoutbaum generiert, der am ende in das Zielformat umgesetzt wird. Der FO-Baum und der verfeinerte FO-Baum unterscheiden sich nur an wenigen stellen und es ist daher nicht nötig den verfeinerten FO-Baum zusätzlich zu erzeugen

Beim FOP wurde dieser Ablauf etwas modifiziert um die Effizienz zu steigern. Um den Speicherplatzbedarf bei großen Dokumenten zu reduzieren, wird immer eine Seite komplett verarbeitet, ausgegeben und dann sofort wieder aus dem Speicher entfernt. Um dies zu erreichen wird ein SAX (Simple API for XML Processing) Parser verwendet, da ein alternativer DOM (Document Object Model) Parser das gesamte Dokument auf einmal in den Speicher lädt. Da die Schritte, die in der Refinement-Phase durchgeführt werden, recht einfach sind, werden diese bereits beim Parsing integriert. Dadurch entsteht nach dem Parsing direkt der verfeinerte FO-Baum und die Pipeline verkürzt sich um eine Stufe und einen Zwischenbaum. Aus dem verfeinerten FO-Baum wird dann der Layout-Baum konstruiert. Sobald eine Seite komplett ist (dies kann erst beim Layout entschieden werden) wird diese gerendert, d.h. in den Ausgabestrom geschrieben in dem jeweiligen Ausgabeformat. Danach werden alle Knoten aus beiden Bäumen (FO-Baum, Layout-Baum) entfernt, die durch diese Seite komplett verarbeitet wurden. FOP ist durch diese Maßnahmen in der Lage, trotz der Java Implementierung, ganze Bücher zu verarbeiten und dies mit einem moderaten Speicherverbrauch.

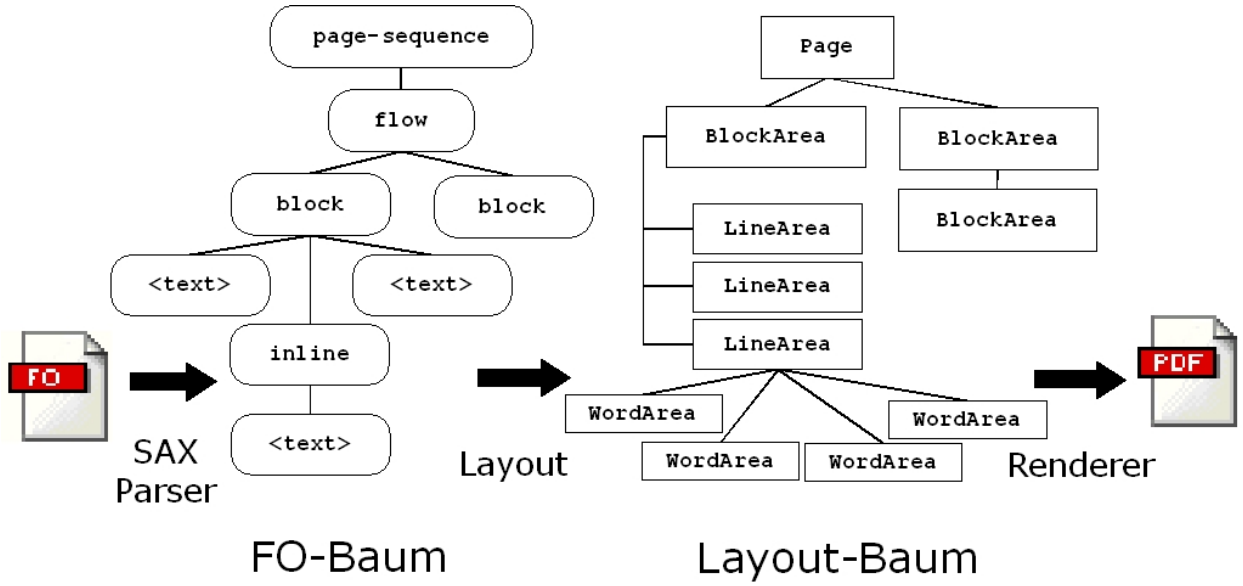

Fig. 4. Bei FOP gibt es nur zwei explizite Bäume. Den FO-Baum und den Layoutbaum. Es werden für beide Bäume typische Elemente und ihre Struktur dargestellt

4.3 Erweiterung von FOP

Die FEX Erweiterung besteht aus einem jar-Archiv mit den Java Klassen und einer Batch-Datei für den Startvorgang. Das FOP System wird nicht verändert. Die Integration von FEX geschieht über abgeleitete Klassen des FOP Systems. Dadurch muss keine modifizierte FOP Version benutzt werden, sondern die bestehende Installation kann weiter verwendet werden. Nur falls man FEX auch benutzten will wird die mitgelieferte Batch-Datei zum Start verwendet, wodurch sich das FEX System in die FOP Pipeline einklinkt.

Wie bereits erwähnt bringt FOP ein paar Erweiterungen mit, die von den Entwicklern selbst geschrieben wurden. Analog zu diesen wird der FEX-Namensraum (http://www.fzi.de/2004/XSL/FormExtension) bei FOP registriert, zusammen mit allen Elementen und ihren Attributen. Dies geschieht über eine Java Hashtabelle in der für jedes Element und jedes Attribut ein Bezeichner zusammen mit einem *Maker* abgelegt wird. Ein *Maker* ist hier eine kleine Klasse, die ein entsprechendes Element bzw. Attribut erzeugt. Es handelt sich um eine abstrakte Fabrik (abstract factory) für nur eine einzige Klasse[15]. Dadurch kann FOP beim Einlesen des FO-Dokuments für die Elemente des FEX-Namesraums die entsprechenden FO-Objekte erzeugen. Für jedes FEX Element gibt es eine von *ExtensionObj* abgeleitete Variante, diese wir durch den entsprechenden *Maker* erzeugt. Beim Erzeugen werden die Attribute, von FOP vorgegebene Klassen z.B. Zeichenkettenttribute oder Längenattribute, extrahiert und teilweise Konsistenzprüfungen vorgenommen. Jedes *decl*-Element braucht ein *id*-Attribut und jedes Element im Text braucht ein *ref*-Attribut dessen Wert auf ein passendes *decl*-Element verweist. Falls eine Prüfung fehlschlägt wird eine entsprechende Fehlermeldung ausgegeben, der Prozess wird aber nicht abgebrochen. Spezielle Verfeinerungsschritte sind für die Formularerweiterung nicht nötig. Zu diesem Zeitpunkt ist die Logikstruktur der Formulare des FO-Dokuments bereits vollständig und wird nicht mehr verändert. Sie dient nur noch zum Nachschlagen der logischen Zusammenhänge. Es wird keine Ausgabe direkt aus der Logikstruktur gewonnen.

Wenn FOP die Elemente eines Blocks eingelesen hat, so beginnt es mit dem Layout, dazu wird bei jedem FO-Objekt die *layout* Methode aufgerufen. Dies geschieht in top-down Form und sorgt dafür das Knoten, die weiter oben in der FO-Hierarchie liegen zuerst ihre entsprechenden Layout-Baum Objekte erzeugen, an die sich ihre Kinder mit ihren Layout-Baum Objekten anheften. Wenn die FEX Objekte zum Layout kommen, erfragen sie von ihren übergeordneten Objekten die Parameter, die sie von dort übernehmen wollen, wie etwa Schriftart, Farbe etc. Dann integrieren sie sich in die momentan aktuelle Zeile des Layout-Baums als ein abgeleitetes *InlineArea* Objekt, das hierfür erzeugt wird und beanspruchen damit einen bestimmten Platz in dieser Zeile. Sollte der Platz in dieser Zeile für das Steuerelement nicht ausreichen, so wird eine neue Zeile angefangen. Bei besonders hohen Steuerelementen wie dem Kommentarfeld oder dem Mehrfachlistenauswahlfeld wird die Höhe der Zeile entsprechend angepasst. Es besteht eine 1 zu 1 zu 1 Zuordnung zwischen den FEX-Elementen im FO-Dokument, den FEX Objekten im FO-Baum und den *InlineArea* Objekten im Layout-Baum.

Wenn FOP erkennt, dass eine komplette Seite zusammengekommen ist, löst es das Rendern aus. Dabei wird bei jedem Objekt im Layoutbaum top-down die *render* Methode aufgerufen und die ein Renderer Interface übergeben wird. Jedes FEX Layout-Objekt schaut ob es sich beim Renderer um einen speziell erweiterten *PDFRenderer* für FEX handelt. Ist dem so, so wird auf ein *ExtendedPDFRenderer* Interface gecasted und die entsprechende Render-Methode aufgerufen, wie etwa *renderButton* oder *renderEdit*. Zudem wird über das alte Render-Interface noch ein Freiraum gerendert, der der Größe des Steuerelements entspricht, dadurch wird zum einen das Textlayout im PDF-Renderer darauf hingewiesen, dass an der Stelle des Steuerelements kein Text stehen kann, zum anderen wird beim Rendern auf andere Formate als PDF anstatt der Steuerelemente einfach ein Freiraum dargestellt. (Eine bessere Lösung für nicht PDF Formate folgt in 4.6). Der Renderer erzeugt für die Steuerelemente entsprechende PDF-Objekte. Darüber hinaus sind noch einige weitere Objekte zu erzeugen, z.B. für die Gruppierung von Umschaltfeldern, der Gruppierung aller Steuerelemente eines Formulars, für auszulösende Aktionen und für Zeichenanweisungen, die das Aussehen von Steuerelementen bestimmen, z.B. für Umschaltfelder und Auswahlfelder. Der PDF Renderer trägt die PDF Objekte der Formulare auch an den entsprechenden Stelle im PDF Dokument ein.

Es wurden hauptsächlich neue Klassen geschrieben, die sich in die verschiedenen Strukturen integrieren: FO-Objekte für den FO Baum, InlineArea-Objekte für den Layout-Baum und PDF-Objekte für die PDF-Erzeugung. Bei den PDF Objekten wurden auch ein paar komplett neu geschrieben für die es davor keine Vorlagen gab. Hier zu zählen *PDFFormAction* für die Absende- und Rücksetztaktionen und verschiedene Klassen ausgehend von *PDFAppearanceStream*, die Zeichenanweisungen enthalten und die Darstellung der Steuerelemente beschreiben.

4.4 Implementierungsprobleme und Lösungen

Das erste Problem, dass sich ergab war, dass, um die Beziehung zwischen einem Steuerelement und seiner Deklaration im Logikteil herzustellen, man die über den *ref*-Wert nach der Deklaration mit der gleichen *id* suchen muss. Dabei ist das Suchen nicht das Problem, sondern der Zugriff auf die Logikstruktur aus den einzelnen FEX Objekten heraus. Das Aufsteigen im FO-Baum bis zur Wurzel und dann das wieder Absteigen bis zum *fex:forms* Element wäre umständlich und ineffizient. Da es nur ein einziges *fex:forms* Element im FO-Dokument gibt, wurde das zugehörige FEX Objekt zum Singleton gemacht. Über eine statische getInstance Methode kann man nun von überall auf die Logikstruktur zugreifen. Leider dient das Singleton

hier mehr als Ersatz einer globalen Variable als irgend einem anderen Zweck.

Das Integrieren neuer FO-Objekte für FEX war relativ einfach, da es bereits ein Erweiterungssystem gab. Jedoch wurde der Renderer und andere Teile von FOP um weitere Methoden erweitert um die Erweiterungen überhaupt erst zu ermöglichen. Die Erweiterungen wurden also eigentlich fest in FOP integriert und sind somit keine echten Erweiterungen. Um die FEX-Erweiterung zu integrieren, mussten deshalb einige bestehenden Klassen erweitert werden. Vor allem der Renderer musst durch einen ersetzt werden, der auch Formularelemente erzeugen kann. Dazu wurde der oben schon erwähnte *ExtendedPDFRenderer* geschrieben, der die meisten Methoden vom ursprünglichen *PDFRenderer* übernimmt und nur die speziellen *renderX* Methoden hinzufügt. Um jedoch FOP anzuweisen auch diesen neuen Renderer zu benutzen, mussten die *Fop*-, *CommandLineStarter*- und *CommandLineOptions*-Klassen durch abgeleitete Versionen ersetzt werden, da im *CommandLineStarter* der Renderer erzeugt wird, der *CommandLineStarter* wird aber in *CommandLineOptions* erzeugt, was wiederum in der *Fop*-Klasse erzeugt wird. Die Entwickler von FOP haben an keiner Stelle die Möglichkeit vorgesehen einen anderen Renderer zu spezifizieren, so das diese drei Klassen durch leicht modifizierte ersetzt wurden.

Beim erzeugen des Layout-Baums werden einige Attribute der Umgebung übernommen. Bei normalem Text geschieht dies automatisch über ein vom übergeordneten Inline-Objekt übergebenen *FontState*. Die FEX-Layoutobjekte werden von den Inline-Objekten jedoch nicht wie Text behandelt, obwohl sie auch den aktuellen *FontState* benötigen. Als Folge bekommen sie einen *FontState* vom übergeordneten Block übergeben, der unter Umständen nicht dem *FontState* entspricht, der innerhalb des Inline Objekts gelten soll. Um nicht auch noch die *Inline*-Klasse zu modifizieren, werden alle Attribute, die die Schrift betreffen aus dem entsprechenden Inline Element des FO-Baums extrahiert, d.h. *font-family*, *font-style*, *font-weight*, *font-size*, *font-variant* und *letter-spacing*. Mit diesen Werten wird dann ein neuer *FontState* konstruiert, der die gewünschten Eigenschaften hat.

Das PDF Format besitzt auch einige Tücken, die das Erzeugen von PDF Dateien erschweren. Bei Formularelementen ist dies insbesondere die Tatsache, das jedes Steuerelement einen Zeichenstrom mitliefern muss, der das Steuerelement in seiner unveränderten Form darstellt. Wenn man ein Steuerelement verändert oder ein Eingabefeld markiert, übernimmt der Acrobat Reader die Darstellung der Steuerelemente und zeichnet diese nach Parametern, die in den PDF Objekten angegeben sind. Verlässt man jedoch das Eingabefeld wieder ohne etwas zu verändern so wird wieder der mitgelieferte Zeichenstrom dargestellt. Wird der Text jedoch verändert und danach auf seinen ursprünglichen Wert zurückgesetzt oder zurück verändert, so stellt der Acrobat Reader die Steuerelement wieder selbst dar. Obwohl der Inhalt mit dem ursprünglichen Inhalt übereinstimmt, wird nicht der mitgelieferte Zeichenstrom verwandt. Warum dieser Zeichenstrom nötig ist obwohl der Acrobat Reader in der Lage ist diese selbständig darzustellen, ist der PDF 1.5 Spezifikation nicht zu entnehmen. Es wurden die Zeichenströme für alle Steuerelement entwickelt, jedoch war es sehr schwer die Darstellung des Acrobat Readers in die Zeichenströme umzusetzen, weil die Darstellung in der Spezifikation nur zum Teil beschrieben wurden. Es war nur durch Ausprobieren und Vergleichen möglich, Linienstärken oder Abstände zu bestimmen. Bestimmte Werte die von der Schriftart abhängen wurden nur für eine Schriftart und Größe bestimmt und müssten für andere Schriftarten ebenfalls bestimmt werden. Damit die Darstellung in den Zeichenströmen mit der Acrobat Reader Darstellung unter allen Schriftarten übereinstimmt wäre sehr viel „Forschung“ nötig. Um dieses Problem zu beseitigen, wurde eine

Rücksetztaktion in jedes PDF Dokument integriert, die beim Laden des Dokuments alle Formulare zurücksetzt. Damit werden die Steuerelemente vom Acrobat Reader als verändert betrachtet und durch diesen auch dargestellt. Die eigenen Zeichenströme kommen nur bei den Steuerelementen zum Einsatz, die der Acrobat Reader nicht selbst darstellt, dazu gehören alle Steuerelemente die keinen Text darstellen, d.h. Umschaltfelder, Auswahlfelder und Schaltflächen. Dadurch wird leider die Fähigkeit die Formulare mit Musterdaten zu versehen eingebüßt. Da die Formulare am Anfang zurückgesetzt werden, spielt das *value* Attribut aller *decl*-Elemente keine Rolle mehr. Nur noch das *default-value* Attribut zählt.

Die Reduzierung des Speicherverbrauchs bei FOP hat einen für die Erzeugung von Formularelementen bei PDF-Dateien unpraktischen Nebeneffekt. Um Speicher zu sparen, werden die Schriftarten, die im Dokument verwendet werden, erst beim Abschließen des Dokuments im PDF-Dokument vermerkt. Dazu wird ein entsprechendes PDF-Objekt für jede Schriftart eingefügt. Bei normalen Zeichenströmen wie sie für die Darstellung der Seiten verwendet werden, werden die verwendeten Schriftarten über Bezeichner angesprochen. Bei Formularelementen müssen die PDF-Objekte der verwendeten Schriftarten referenziert werden. Diese existieren aber während dem Erzeugen der Seiten und damit der Steuerelemente noch nicht. Die Schriftartenobjekte werden nicht dann erzeugt wenn sie gebraucht werden, sonder erst ganz am Schluss. Um dieses Problem zu beheben, werden dann, wenn die Schriftart benötigt wird, eine entsprechende Seriennummer reserviert.[7] Über diese Nummer wird die Schriftart in den Steuerelementen referenziert. Wenn am Ende die Schriftarten Objekte erzeugt werden, so werden die reservierten Seriennummern verwendet. Dies ist eigentlich eine Verletzung der Kapselung, der PDF Objekte, da eigentlich nur sie selbst mit ihren Seriennummern arbeiten und anderen Objekte Referenzen über eine *referencePDF*-Methode bekommen. Man hätte vielleicht die Erzeugung der PDF-Schriftartenobjekte verändern können, doch handelt es sich bei dem Schriftartensystem um ein relativ komplexes System, so das zu Gunsten dieses unschönen Workarounds entschieden wurde.

4.5 Einschränkungen der Implementierung

Die Fex-Spezifikation wurde ohne Berücksichtigung der nachfolgenden Implementierung entwickelt. Erst beim Implementieren ergaben sich Probleme, die sich nur nur durch erheblichen Mehraufwand hätten lösen lassen. Darunter fallen die eigenen Zeichenströme der Steuerelemente und leider auch die Validierung der Eingabefelder mittels des regulären Ausdrucks. Das PDF-Format unterstützt keine Validierung der Eingabefelder über das vorhandene Formularsystem. PDF-Dateien können jedoch ECMA-Script enthalten, das wie bei HTML über ein DOM Zugriff auf sämtliche Elemente eines Dokuments besitzt [16]. Hiermit wäre eine Validierung, die bei einem Verändern-Ereignis ausgelöst wird, möglich. Jedoch ist ECMA-Script (AKA JavaScript) kein Java, so dass das java.util.regex API nicht zur Verfügung steht und damit eine Umsetzung komplizierter wird. Über ECMA-Script wäre auch ein Setzen des Initialwertes möglich, so dass die Musterwerte genutzt werden könnten und trotzdem der Acrobat Reader die Darstellung übernimmt. Aus Zeit- und Komplexitätsgründen wurde die Nutzung von ECMA-Script in PDF-Dateien weg gelassen, wäre aber ein wichtiger Punkt bei einer Fortführung der FEX-Erweiterung.

Es war am Anfang des Projekts angedacht auch den AWT-Renderer von FOP um die Unterstützung von FEX zu erweitern. Der Hauptgrund warum dies nicht zustande gekommen

7 Jedes Objekt in einem PDF-Dokument bekommt eine Seriennummer. Die Position eines Objekts wird in einer Tabelle am Ende eines PDF-Dokuments abgelegt. PDF-Objekte referenzieren sich untereinander meistens über diese Seriennummern.

ist, ist der, dass die Erweiterung des AWT-Renderers sogar mehr Arbeit bedeutet hätte als beim PDF-Renderer. Das liegt daran, dass der AWT-Renderer die einzelnen Seiten in eine Bitmap rendert und nicht wie beim PDF-Renderer eine Sammlung von Objekten erzeugt. Die Struktur und die Positionen der Steuerelemente hätten komplett selbst entwickelt werden müssen, was den Aufwand beträchtlich erhöht hätte.

4.6 Weitere Komponenten

Um die FEX-Erweiterung auch ohne FOP und die Erweiterung dafür nutzen zu können, wurde eine XSL Transformation entwickelt, die ein mit FEX erweitertes FO-Dokument in ein reines FO-Dokument umwandelt. Dazu werden die FEX-Steuerelemente in reine Textform umgewandelt. Für diese Umwandlung gibt es zwei Möglichkeiten: Zum einen kann das unausgefüllte Formular erzeugt werden, das dann auch auf Papier ausgedruckt ausgefüllt werden kann. Zum anderen können die bereits eingetragenen Werte (value Attribute) in das Formular eingefügt werden, da durch ist es möglich dem Nutzer nach dem Ausfüllen das Ergebnis in einer nicht mehr veränderbaren Fassung zu geben. Um dies zu Automatisieren wäre eine Pipeline nötig, die die abgesendeten Daten in das ursprünglich FO-Dokument einträgt, dann mit Hilfe der oben beschriebenen XSL Transformation in ein reines FO-Dokument umwandelt und aus diesem dann einen Ausdruck oder ein PDF-Dokument generiert.

4.7 Erfahrungen

Das schwierigste an der Implementierung war zu verstehen, wie die verschieden Objekte von FOP und des PDF-Formats zusammenarbeiten, um dann diese zu Erweitern. Bei einer nochmaligen Implementierung würde man die Klassen der Erweiterung wohl etwas anders gestalten, indem Funktionalität, die in mehreren Klassen vorkommt, in eine gemeinsame Basisklasse verschoben wird. Wäre wohl auch durch Refactoring möglich. Viele Entwurfsentscheidungen wurden aber durch die gegebene Struktur von FOP und PDF bestimmt, so dass es hier kaum Freiheiten beim Entwurf gab. Eine Erweiterung eines anderen FO-Prozessors wäre vielleicht auf eine elegantere Weise möglich.

Der Renderer erfüllt jedoch die meisten Anforderungen und kann beliebige Formulare mit allen in FEX enthaltenen Steuerelementen erzeugen. Die Entscheidung, über eine automatische Rücksetztaktion die Darstellung durch den Acrobat Reader zu erzwingen, war schmerzlich, da die Möglichkeit, Vorgaben zu integrieren, eine der Fähigkeit von FEX ist.

5 Zusammenfassung und Ausblick

Es wurde eine Erweiterung für XSL-FO entwickelt, die es ermöglicht Formulare zu spezifizieren, die in Umfang und Fähigkeiten den Formularen von HTML entsprechen. Die Strukturierung in Logikteil und Steuerelementbeschreibung, ermöglicht eine klare Trennung zwischen diesen Gebieten und führt zu einer einfachen Integration in XSL-FO.

Um die entwickelte Erweiterung zu testen, wurde der FO-Prozessor FOP um die Verarbeitung der Erweiterung ergänzt. Dabei konnten fast alle Fähigkeiten der FEX Spezifikation implementiert werden. Der modifizierte FO-Prozessor ist in der Lage PDF-Dateien mit Formularelementen zu erzeugen. Diese können im Acrobat Viewer über HTTP POST und GET versandt werden, genau so wie bei HTML Formularen über einen Browser.

Die Erweiterung und die Implementierung stellen einen soliden Ausgangspunkt für eine Weiterentwicklung von FEX dar. Die Grundeigenschaften von Web-Formularen sind erreicht und man könnte Eigenschaften wie clientseitige Valididerung oder weitere Steuerelemente der Erweiterung hinzufügen. Diese Ideen sind in XForms bereits vorgesehen, jedoch müssten Erweiterungen für XSL-FO unter dem Gesichtspunkt der Spezifikation für Printmedien und weniger der Interpretation eines Browsers wie bei XForms entwickelt werden. Eine Erweiterung von FEX in ein so umfassendes System wie XForms würde jedoch die Komplexität erhöhen und damit den Einsatz vor allem für Leute, die sich hauptsächlich mit dem Layout beschäftigen, erschweren. Die FEX-Erweiterung hat im jetzigen Zustand noch einen recht simplen Charakter, was man als positive Eigenschaft werten kann und gerade deshalb bewahren will.

Referenzen

1. World Wide Web Consortium: XSL 1.1 (2003)
2. Kay, Michael: XSLT - Programmer's Reference, 2nd Edition (2001)
3. World Wide Web Consortium: XForms 1.0 (2003)
4. MSDNTV: Episode on December 18th (2003)
5. The .NET Show: Longhorn Avalon, May 10th (2004)
6. Microsoft: http://msdn.microsoft.com/longhorn/ (2004)
7. Bishop, Judith and Horspool, Nigel: http://views.cs.up.ac.za/ (2004)
8. Bishop, Judith and Horspool, Nigel: C# Concisely, Addison-Wesley (2003)
9. Simpson, John E.: Just XSL, 2nd Ed. (2002)
10. Adobe Systems Incorporated: PDF Reference - Version 1.5 (2003)
11. Adobe Systems Incorporated: XFA Specification - Draft - Version 2.1 (2004)
12. Adobe Systems Incorporated: http://www.adobe.com/enterprise/xml.html (2004)
13. Sun Microsystems, Inc.: Abstract Window Toolkit (AWT) http://java.sun.com/j2se/1.4.2/docs/guide/awt/ (2002)
14. apache.org: http://xml.apache.org/fop/ (2004)
15. Gamma, Erich et al.: Design Patterns: Elements of Reusable Object-Oriented Software (1995)
16. Adobe Systems Incorporated: Acrobat JavaScript Scripting Reference (2004)